

MULTIMEDIA FORENSICS USING METADATA

by

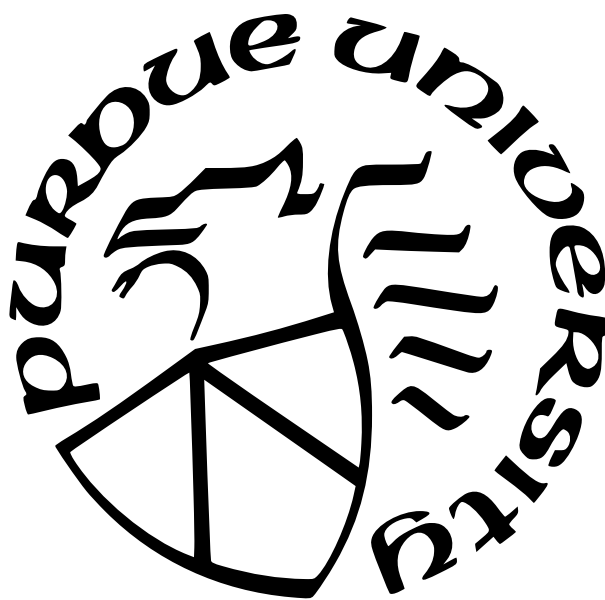
Ziyue Xiang

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy



School of Electrical and Computer Engineering

West Lafayette, Indiana

May 2024

**THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL**

Dr. Edward J. Delp, Chair

School of Electrical and Computer Engineering

Dr. Amy R. Reibman

School of Electrical and Computer Engineering

Dr. Fengqing M. Zhu

School of Electrical and Computer Engineering

Dr. Mary L. Comer

School of Electrical and Computer Engineering

Approved by:

Dr. Milind Kulkarni

ACKNOWLEDGMENTS

I would like to thank my doctoral advisor Professor Edward J. Delp, who accepted me into the VIPER lab during trying times worldwide. I would like to thank Professor Amy R. Reibman, Professor Fengqing M. Zhu, and Professor Mary L. Comer for their support in composing this dissertation. I would like to thank Professor Paolo Bestagini for his support throughout my Ph.D. program. I would like to express my gratitude to János Horváth, Sriram Baireddy, Amit K. S. Yadav, Kratika Bhagtani, as well as all other former and current VIPER lab members for the support and friendship since the beginning of my Ph.D. program. I would like to thank Professor Daniel E. Acuña for his guidance throughout my masters studies, which granted me the enthusiasm and determination to pursue a Ph.D. degree. Finally, I would like to thank my family for their unconditional love and continued support.

This material is based on research sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under agreement number FA8750-20-2-1004. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA and AFRL or the U.S. Government.

TABLE OF CONTENTS

LIST OF TABLES	11
LIST OF FIGURES	13
NOMENCLATURE	15
ABSTRACT	18
1 INTRODUCTION	19
1.1 Forensic Analysis of Video Files Using Metadata	19
1.2 Forensic Analysis and Localization of Multiply Compressed MP3 Audio Using Transformers	19
1.3 H4VDM: H.264 Video Device Matching	20
1.4 MTN: Forensic Analysis of MP4 Video Files Using Graph Neural Networks	21
1.5 Extracting Efficient Spectrograms From MP3 Compressed Speech Signals for Synthetic Speech Detection	21
1.6 Contributions Of This Work	22
2 FORENSIC ANALYSIS OF VIDEO FILES USING METADATA	23
2.1 Introduction	23
2.2 Related Work	24
2.3 Proposed Approach	25
2.3.1 An Overview of Our Approach	25
2.3.2 Video Metadata	26

2.3.2.1	Parsing <code>ilst</code> Data	28
2.3.2.2	Parsing XML Data	29
2.3.3	Track and Type Aware Feature	30
2.3.4	Feature Selection	33
2.3.5	Dimensionality Reduction and Classification	34
2.3.5.1	Multi-class problems	34
2.3.5.2	Two-class problems	35
2.4	Experiments and Results	35
2.4.1	Scenario 1: Brand Attribution	36
2.4.1.1	Close-set scenario	36
2.4.1.2	Blind scenario	36
2.4.2	Scenario 2: Manipulation Tool Identification	37
2.4.3	Scenario 3: Social Network Attribution	39
2.4.4	Scenario 4: Manipulation Detection	40
2.4.4.1	Manipulation detection on videos files from social networks . . .	40
2.4.4.2	Manipulation detection on local videos	41
2.5	Conclusion	42
2.6	Supplementary Materials	43
2.6.1	Spectral Clustering	43
2.6.2	Linear Discriminant Analysis (LDA)	45

3	FORENSIC ANALYSIS AND LOCALIZATION OF MULTIPLY COMPRESSED MP3 AUDIO USING TRANSFORMERS	47
3.1	Introduction	47
3.2	Background	48
3.2.1	MP3 Compression	48
3.2.2	Transformer Neural Networks	49
3.3	Multiple MP3 Compression Localization	50
3.3.1	Problem Formulation	50
3.3.2	Proposed Method	51
3.4	Experiments and Results	54
3.4.1	Dataset	54
3.4.2	Preparing an MP3 file For Training	55
3.4.3	Hyperparameters and Training	56
3.4.4	Results	56
3.5	Conclusions	57
4	H4VDM: H.264 VIDEO DEVICE MATCHING	59
4.1	Introduction	59
4.2	Background	61
4.2.1	Related Work	61
4.2.2	H.264 Video Compression	62

4.2.3	Vision Transformers	63
4.3	Proposed Method	64
4.3.1	H4VDM Feature Extractor	65
4.3.1.1	I-Proc.	66
4.3.1.2	DF-Proc.	66
4.3.1.3	FT-Proc.	66
4.3.1.4	M-Proc.	67
4.3.1.5	L-Proc.	67
4.3.2	Similarity Score and Loss Function	68
4.4	Experiments and Results	69
4.4.1	Dataset Generation	69
4.4.2	Parameter Initialization and Training	72
4.4.3	Model Size Selection	73
4.4.4	Results	73
4.4.4.1	Results on Datasets D1–D4.	73
4.4.4.2	Results on Datasets D5–D7.	74
4.5	Conclusion	74
5	MTN: FORENSIC ANALYSIS OF MP4 VIDEO FILES USING GRAPH NEURAL NETWORKS	78
5.1	Introduction	78

5.2	MP4 Tree Network (MTN)	79
5.2.1	MP4 Tree Transformation	80
5.2.1.1	MP4 Tree Restructuring	81
5.2.1.2	String Processing	82
5.2.2	Node Embedding Network (NEN)	83
5.2.2.1	Number Encoder	84
5.2.2.2	SOL Encoder	86
5.2.2.3	Graph Analysis Module (GAM)	86
5.2.2.3.1	Node tag encoding.	87
5.2.2.3.2	GAtN	87
5.2.2.4	Self-Supervised NEN Pretraining	89
5.2.3	Graph Convolution and Graph Pooling	90
5.3	Data Augmentation	91
5.4	Experiments and Results	92
5.4.1	NEN Pretext Tasks	92
5.4.2	Forensics Analysis Using MTN	93
5.4.2.1	The Robustness of MTN	95
5.4.3	Ablation Study	97
5.5	Conclusion	97
5.6	Supplementary Materials	98

5.6.1	The Tree Restructuring Algorithm	98
5.6.2	String Processing	98
5.6.3	Graph Pooling (GP) Readout	100
5.6.4	Model Size of the Node Embedding Network (NEN)	101
5.6.5	Model Architecture	101
6	EXTRACTING EFFICIENT SPECTROGRAMS FROM MP3 COMPRESSED SPEECH SIGNALS FOR SYNTHETIC SPEECH DETECTION	102
6.1	Introduction	102
6.2	Related Work	103
6.3	Background	103
6.3.1	MP3 Compression	104
6.3.2	Quadratic Mirror Filter (QMF)	104
6.3.3	Speech Features	105
6.3.3.1	The Spectrogram	105
6.3.3.2	Mel-Frequency Cepstrum Coefficients (MFCC)	106
6.3.3.3	Constant-Q Transform (CQT)	107
6.4	Efficient Spectrograms (E-Specs)	108
6.4.1	Theoretical Formulation	109
6.4.2	Efficiency Analysis	110
6.4.3	Implementation of Efficient Spectrogram (E-Spec)	112

6.5	Experimental Results	112
6.6	Conclusion	115
7	SUMMARY AND FUTURE WORK	117
7.1	Forensic Analysis of Video Files Using Metadata	117
7.2	Forensic Analysis and Localization of Multiply Compressed MP3 Audio Using Transformers	117
7.3	H4VDM: H.264 Video Device Matching	118
7.4	MTN: Forensic Analysis of MP4 Video Files Using Graph Neural Networks	118
7.5	Extracting Efficient Spectrograms From MP3 Compressed Speech Signals for Synthetic Speech Detection	119
7.6	Contributions Of This Work	119
	REFERENCES	121
	VITA	139
	PUBLICATION(S)	140

LIST OF TABLES

2.1	A list of common keys in <code>ilst</code> boxes [31].	29
2.2	F1-score comparison of device attribution scenario.	37
2.3	F1-score comparison of manipulation tool identification scenario.	38
2.4	F1-score comparison of social network attribution scenario.	39
2.5	Performance evaluation metrics comparison between our approach and previous work. TPR and TNR stand for True Positive Rate and True Negative Rate, respectively. The accuracy score has been balanced.	41
2.6	Comparison of our method with previous works. The balanced accuracy is averaged over 34 folds.	42
3.1	MP3 codec information fields used in our method. Details about each field can be obtained from [73], [74].	52
3.2	MP3 compression types used in our experiments. The data rate of VBR compression decreases as quality index increases. More details can be obtained from [80].	54
3.3	Performance metrics comparison.	57
3.4	The recall of multiple compression localization of each method against selected last MP3 compression types. CBR compression is denoted by C<bit rate>; VBR compression is denoted by V<quality index>.	58
3.5	The recall of each method against the number of MP3 compression.	58
4.1	The hyperparameters of the vision transformers (ViT) [69] used in H4VDM, as discussed in Section 4.4.3.	67
4.2	The list of devices contained in the VISION dataset [132].	70
4.3	The details of each dataset. #0 and #1 indicates the number of Group of Picture (GOP) pairs with class 0 (different device) and class 1 (same device), respectively.	71
4.4	The best AUC scores of various models on the testing set of Dataset D1.	73
4.5	The testing performance of the H4VDM-B model on datasets D1–D4.	74
4.6	The testing performance of the H4VDM-B model on datasets D5–D7.	75
5.1	The F ₁ -score comparison of the social network attribution task. The performance of “(Local)” category is not available for Yang <i>et al.</i> [137].	95
5.2	The F ₁ -score comparison of the editing tool attribution task.	95
5.3	The balanced accuracy score comparison of the manipulation detection task.	95
5.4	The robustness of random forest and MTN against the evidence removal attack.	97

5.5	The choice of M and the corresponding size of the Node Embedding Network (NEN). .	101
6.1	The number of parameters of each neural network architecture used in our experiments.	113
6.2	The AUC scores (%) of ResNet50 [195] synthetic speech detector given different E-Spec extraction settings.	114
6.3	The EER (%) of ResNet50 [195] synthetic speech detector given different E-Spec extraction settings.	115
6.4	The synthetic speech detection AUC score (%) of speech features computed from decoded speech and E-Spec across different neural network architectures.	115
6.5	The synthetic speech detection EER (%) of speech features computed from decoded speech and E-Spec across different neural network architectures.	116

LIST OF FIGURES

2.1	The structure of our proposed metadata forensic analysis technique.	26
2.2	Illustration of the MP4 file format, where each cell represents one byte. An MP4 file is made up of a series of <i>boxes</i> . Every box has an 8-byte header, where the first 4 bytes store the size of the box in bytes (including the header) as a 32-bit big-endian integer, and the last 4 bytes store the name of the box. The content of a box is either child boxes or binary data. Binary data from different boxes may require distinct decoding methods.	27
2.3	Examples of representing MP4 metadata tree with strings. Node paths are separated with '/', the values of leaf nodes are prefixed with '@', non-ASCII and unprintable characters are shown as hexadecimal codes surrounded by black frames. The metadata tree of any MP4 file can be portrayed by a collection of such strings. . . .	28
2.4	Examples of XML data in MP4 video containers.	30
2.5	Illustration of the vector representation of MP4 metadata. The χ functions help determine the corresponding element of a node or a metadata field in the feature vector.	31
2.6	2D feature distribution and classification boundary for device attribution scenario. .	37
2.7	Example of blind device attribution scenario. The projected samples from the unknown device are shown with white markers with black contour.	38
2.8	2D feature distribution and classification boundary for manipulation tool identification scenario.	39
2.9	2D feature distribution and classification boundary for social network attribution scenario.	40
3.1	The block diagram of an MP3 encoder.	49
3.2	The block diagram of the proposed method, which analyzes L frames at a time. Each circular node (\odot) represents a MP3 codec parameter vector whose corresponding frame is shown by the number inside. Each diamond node (\diamond) represents a vector associated with the feature vector z'_l . Each rectangular node ($\boxed{c_l}$) represents a vector associated with the class token c_l . Different groups of vectors are shown in different colors.	51
3.3	An illustration of our dataset generation method on a segment of 40 frames (4 slices). The corresponding label for this segment y is shown at the bottom.	55
4.1	The block diagram of our proposed H.264 Video Device Matching (H4VDM) method. The details of the H4VDM feature extractor are described in Section 4.3.1. Note the input GOP information includes the decoded frame and the coding parameters. 64	
4.2	The block diagram of the H4VDM feature extractor.	65

4.3	The similarity score function.	69
4.4	The testing accuracy score matrix of device index pairs on datasets D1–D4.	76
4.5	The testing accuracy score matrix of device index pairs on datasets D5–D7.	77
5.1	The block diagram of MP4 Tree Network (MTN).	79
5.2	The illustration of the tree structure of MP4 video files.	81
5.3	The restructured MP4 tree derived from the tree illustrated in Figure 5.2. For each node, the tag is shown in the circle and the data is shown in the rectangle.	82
5.4	The block diagram of NEN.	84
5.5	The image of $\text{symlog}(x)$	85
5.6	The training performance of pretext tasks using different (M, γ) settings.	92
5.7	The validation performance of different training settings vs. training steps.	98
6.1	The block diagram of the MP3 signal decoding process, which helps illustrate the our efficient spectrogram extraction method. MP3 compression splits the signal into 32 frequency bands using a band-pass analysis filterbank. Extracting the spectrogram from the frequency bands is more efficient because the high frequency bands are redundant; the length of each band is shorter; and the synthesis filterbank is not involved.	103
6.2	Illustration of the Mel filterbank for DFT coefficients. Note that both the frequency bins and filter magnitude are changing based on the characteristics of the human hearing system.	107
6.3	Magnitude response of the band-pass analysis filterbank used in MP3. f_N denotes the Nyquist frequency of the signal.	108
6.4	The interpolation window W_w	111
6.5	The percentage of time used by each step in the MP3 decoding process. The results were averaged after 100 repeated experiments using the <code>minimp3</code> [191] decoder.	112
6.6	Spectrogram and E-Specs extracted from an example speech signal.	114

NOMENCLATURE

AAC Advanced Audio Codec. 47

AUC Area Under the Receiver Operating Characteristic. 69, 72, 73, 74, 113, 114

CNN Convolutional Neural Network. 52, 57, 90

CQT Constant-Q Transform. 21, 102, 103, 105, 107, 108, 113, 116, 119

DCT Discrete Cosine Transform. 106, 107

DFT Discrete Fourier Transform. 106, 109, 110

DNN Deep Neural Network. 48

EER Equal Error Rate. 113, 114, 115

E-Spec Efficient Spectrogram. 9, 12, 14, 21, 102, 103, 112, 113, 114, 115, 116, 119

FFT Fast Fourier Transform. 48, 111, 112

GA Graph Attention. 87, 88, 92

GAM Graph Analysis Module. 83, 86, 89, 101

GAtN Graph Attention Network. 86, 87, 92, 101

GC Graph Convolution. 79, 90, 94, 100, 101

GL Graph Laplacian. 44

GNN Graph Neural Network. 18, 21, 22, 78, 79, 80, 81, 82, 83, 97, 100, 118, 120

GOP Group of Picture. 11, 25, 60, 62, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 118

GP Graph Pooling. 79, 90, 91, 94, 100, 101

HHS Human Hearing System. 48, 49

IMDCT Inverse Modified Discrete Cosine Transform. 104

LDA Linear Discriminant Analysis. 34, 45

MDCT Modified Discrete Cosine Transform. 47, 48, 49, 103, 104

MFCC Mel-Frequency Cepstrum Coefficients. 21, 102, 103, 105, 106, 108, 113, 116, 119

MLP Multilayer Perceptron. 53

MSA Multihead Self Attention. 50, 63, 86, 87, 88, 92

NEN Node Embedding Network. 12, 14, 79, 83, 84, 89, 90, 92, 93, 94, 97, 101

NLP Natural Language Processing. 83, 84, 89

p.p. percentage points. 102, 115, 119

PRNU Photo Response Non-uniformity. 59, 61

QMF Quadrature Mirror Filter. 104, 105

QP Quantization Parameter. 62, 63, 66, 67, 68

ROC Receiver Operating Characteristic. 113

SA Self Attention. 49, 50, 63, 87, 88, 91

SOL String Object List. 80, 83, 86, 87, 89, 90, 92, 101

SSL Self-Supervised Learning. 18, 21, 22, 79, 83, 89, 90, 97, 118, 119, 120

STFT Short Time Fourier Transform. 21, 102, 105, 106, 107, 108, 109, 110, 111, 112, 113, 116, 119

SVM Support Vector Machine. 78, 79

TNR True Negative Rate. 73

TPR True Positive Rate. 73

VDI Video Device Identification. 59, 60, 61

VDM Video Device Matching. 60, 61, 62, 63, 64, 74, 75, 118

VFM Video Forensics Method. 78, 94, 96

ViT Vision Transformer. 63, 65

XMP Extensible Metadata Platform. 29

ABSTRACT

The rapid development of machine learning techniques makes it possible to manipulate or synthesize video and audio contents while introducing nearly undetectable artifacts. Most media forensics methods analyze the high-level data (e.g., pixels from videos, temporal signals from audios) decoded from compressed media data. Since media manipulation or synthesis methods usually aim to improve the quality of such high-level data directly, acquiring forensic evidence from these data is becoming increasingly challenging. In this thesis, we focus on media forensics techniques using the metadata in digital media, which includes metadata and coding parameters in bitstreams. Since many media manipulation and synthesis methods do not attempt to hide metadata traces, metadata based forensic analysis can lead one to more forensic evidence. In this thesis, we first present a video forensics technique using the metadata embedded in MP4/MOV video sequences. Our proposed method achieved high performance in video manipulation detection, source device attribution, social media attribution, and manipulation tool identification on publicly available datasets. Second, we present a transformer neural network based MP3 audio forensics technique using low-level codec information. Our proposed method can localize multiple compressed segments in MP3 files. The localization accuracy of our proposed method is higher compared to other methods. Third, we present an H.264-based video device matching method. This method can determine if the two video sequences are captured by the same device even if the method has never encountered the device. Our proposed method achieved good performance on a publicly available video forensics dataset containing 35 devices. Fourth, we present a Graph Neural Network (GNN) based approach for the analysis of MP4/MOV metadata trees. The proposed method is trained using Self-Supervised Learning (SSL), which increased the robustness of the proposed method and makes it capable of handling missing/unseen data. Fifth, we present an efficient approach to compute the spectrogram feature with MP3 compressed audio signals. The proposed approach decreases the complexity of speech feature computation by $\sim 77.6\%$ and saves $\sim 37.87\%$ of MP3 decoding time. The resulting spectrogram features lead to higher synthetic speech detection performance.

1. INTRODUCTION

1.1 Forensic Analysis of Video Files Using Metadata

Digital video has become one of the major channels for the general public to acquire information. With the advancement of computer technology and machine learning, it is easier to manipulate video data or synthesize video content than ever before. The quality of such manipulated or synthesized video sequences has improved to a point where it can be difficult for human eyes to identify problems in them. Therefore, the digital media forensics community has been developing methods based on statistical analysis, signal processing, and machine learning to help verify the authenticity of video sequences. Most of the digital video forensics methods are based on pixel information. Due to the high dimensionality of the pixel information, these video forensics methods tend to be sophisticated and computationally intensive. Because of the focus on pixel information, many counterforensics methods are trying to attack pixel-based video forensics methods, which increases the difficulty of accurate detection in the pixel domain.

In this chapter, we propose a video forensics method using the metadata in MP4/MOV video containers. Since our method only uses metadata information to make decisions, it breaks out of the forensics-counterforensics loop in the pixel domain. Our method can be used for a wide variety of video forensic tasks, including video manipulation detection, video source device attribution, social media attribution, and video manipulation detection. Our proposed approach can reduce the dimensionality of video metadata features to two and generate 2D feature scatter plots and decision boundary graphs for many video forensics tasks. This enables us to gain insights into the distribution of MP4 metadata and make interpretable decisions. Our proposed metadata-based video forensics method runs considerably faster than pixel-based methods. Our experimental results show that many video forensics problems on standard datasets can be solved reliably by looking only at metadata.

1.2 Forensic Analysis and Localization of Multiply Compressed MP3 Audio Using Transformers

MPEG-1 Audio Layer III (MP3) is one of the most popular audio compression standards. MP3 compression leaves artifacts in the compressed audio bitstream. By analyzing the compression artifacts, it is possible to find out what parts of an MP3 audio sequence have undergone multiple

MP3 compression. The inconsistencies in terms of the number of MP3 compressions can help one determine the authenticity of an MP3 audio sequence. In this chapter, we propose a MP3 multiple compression localization method. Given an MP3 signal, our proposed method can distinguish between single compressed temporal segments and multiple compressed temporal segments thereby allowing us to temporally localize where the audio signal may have been spliced. Our method extracts low-level information from the MP3 bitstream such as MDCT coefficients, scalefactors, and Huffman table indices for analysis. The extracted low-level information are processed by a transformer neural network to generate final decisions. Our proposed method localizes multiple compression with a finer granularity compared to existing methods. The experiment results showed that our method had the best performance compared to other approaches and was robust against many MP3 compression settings.

1.3 H4VDM: H.264 Video Device Matching

Video Device Matching (VDM) is the task of determining if two video sequences are captured by the same device. With sample videos from known devices, we can use VDM methods to find the source device of a video sequence. VDM techniques can be useful in forensic investigations. In this chapter, we propose a VDM method for video sequences compressed using the H.264 standard. Given two H.264 video sequences, our proposed method can generate a similarity score, which can be used to determine if the two sequences are captured by the same device. Our method uses the decoded frames and the low-level codec information from the H.264 bitstream to make decisions, which includes frame types, macroblock types, and quantization parameters. Our method can achieve open-set detection, which means it can still work even if the two input video sequences are not seen by the method before. Our proposed method achieved good performance in a three-fold cross validation scheme on a publicly available video forensics dataset containing 35 devices. The experimental results showed that our method demonstrated good VDM performance on unseen devices.

1.4 MTN: Forensic Analysis of MP4 Video Files Using Graph Neural Networks

MP4 video files are stored using a tree data structure. These trees contain rich information that can be used for forensic analysis. In this chapter, we propose MP4 Tree Network (MTN), an approach based on an end-to-end Graph Neural Networks (GNNs) that is used for forensic analysis of MP4 trees. MTN does not use any video pixel data. MTN is trained using Self-Supervised Learning (SSL), which generates semantic-preserving node embeddings for the nodes in an MP4 tree. We also propose a data augmentation technique for MP4 trees, which helps train MTN in data-scarce scenarios. MTN achieves good performance across 3 video forensics tasks on the EVA-7K dataset. We show that MTN can gain more comprehensive understanding about the MP4 trees and is more robust to potential attacks compared to existing methods.

1.5 Extracting Efficient Spectrograms From MP3 Compressed Speech Signals for Synthetic Speech Detection

Many speech signals are compressed with MP3 to reduce data rate. We discover that the design of MP3 compression allows efficient computation of the Short Time Fourier Transform (STFT) of the speech signal compressed by MP3. STFT is a building block of many speech features such as Mel-Frequency Cepstrum Coefficients (MFCC), Constant-Q Transform (CQT) and spectrogram, which are used in a wide range of speech analysis tasks. Our proposed technique can reduce the complexity of STFT computation by $\sim 77.6\%$ and save $\sim 37.87\%$ of MP3 decoding time. It can also bypass the reconstruction artifacts introduced by the MP3 synthesis filterbank, which affects the signal decoded from MP3 compression. We computed the spectrogram features using our proposed efficient computation approach (denoted by Efficient Spectrogram (E-Spec)) and used them for the synthetic speech detection task, in which a detector is asked to determine whether a speech signal is synthesized by computer or recorded from human. We compared the synthetic speech detection performance of E-Spec to speech features extracted from MP3 decoded speech signal using 4 neural network architectures. E-Spec achieved the best synthetic speech detection performance for 3 architectures. E-Spec also achieved the best overall detection performance across architectures. Our proposed technique can be extended to other audio compression methods.

1.6 Contributions Of This Work

In this work, we developed new methods for multimedia forensics using metadata. The main contributions of the work are listed as follows:

- We proposed a video forensics method using the metadata in MP4/MOV video containers, which allows the visualization of video metadata features and achieves good performance in a wide variety of video forensics tasks
- We proposed an MP3 multiple compression localization method using transformer neural networks, which outperforms other methods in terms of both localization granularity and accuracy
- We proposed a video device matching method for H.264 video sequences, which can achieve open-set video device matching with high accuracy
- We proposed a Graph Neural Network (GNN) based method for the analysis of MP4/MOV metadata trees; the GNN is trained using Self-Supervised Learning (SSL), which increases the robustness of the analysis and makes it capable of handling missing/unseen data
- We proposed a technique to compute the spectrogram feature from MP3 compressed signals without fully decoding them; the proposed technique decreases the complexity of speech feature computation by $\sim 77.6\%$ and saves $\sim 37.87\%$ of MP3 decoding time; the resulting spectrogram features lead to higher synthetic speech detection performance

2. FORENSIC ANALYSIS OF VIDEO FILES USING METADATA

2.1 Introduction

The proliferation of easy-to-use video manipulation tools has placed unprecedented power in the hands of individuals. Recently, an Indian politician used deepfake technology to rally more voters [1]. In the original video the politician delivered his message in English; it was convincingly altered to show him speaking in local dialect. Media manipulation methods are also used as tools of criticism and to undermine the reputation of politicians [2]. Such manipulated videos can now be easily generated to bolster disinformation campaigns and sway the public opinion on critical issues.

A wide variety of tools for video forensic analysis have been developed [3]. These tools can be used to attribute a video to the originating device, to reconstruct the past video compression history, and even to detect video manipulations. The most popular video manipulation detection techniques focus on inconsistencies and artifacts in the pixel domain [4]–[7]. As video manipulation detection methods become more sophisticated, video editing techniques continue to improve, leading to a situation where manipulated videos are becoming practically indistinguishable from real videos [8]–[13]. For this reason, detection techniques exploiting pixel-level analysis may fail, while methods that do not use pixel data will increasingly gain importance.

Video forensic techniques not exploiting pixel analysis typically work due to the presence of “metadata” [14], [15]. This is additional embedded information that every video file contains. The metadata are used for video decoding [16] and indicating other information such as the date, time, and location of the video when created. Because video editing tools tend to cause large structural changes in metadata, it is difficult for one to alter a video file without leaving any metadata traces. Therefore, metadata can serve as strong evidence in video forensics tasks.

In this chapter, we leverage the seminal work presented in [15], [17] to investigate the use of metadata for video forensic analysis of the MP4 and MOV video formats, which are among the most popular video wrappers/containers. The MP4 format is used by numerous Android devices, social networks, and digital video cameras [18]–[20]. MOV format is mostly used by Apple devices and is derived from the same ISO standard as MP4 [21]. The design of the MP4 format is based on MOV [22]. The two formats can be parsed in a similar manner, thus we will refer to MP4 containers

hereinafter even if MOV containers are considered. As a result, our approach can analyze a large number of videos in the real world.

In our work, we examine the results of using the metadata in MP4 files for video forensic scenarios, extending the work presented in [17]. More specifically, we describe a metadata extraction method and improve the feature representation format to make metadata-based forensic feature vectors more compact. We employed feature selection techniques to boost the quality of the feature vectors. Finally, we reduced the dimensionality of the feature vectors to two, which allows visualization and classification in 2D space. We show that these feature vectors can be used for a wide variety of video forensic tasks, from source attribution to tampering detection. Compared to other work, our proposed approach can generate 2D feature scatter plots and decision boundary graphs for many video forensics tasks. This feature enables us to gain insights into the distribution of MP4 metadata and make interpretable decisions.

Our experimental results show that many video forensics problems on standard datasets can be solved reliably by looking only at metadata. We also discovered that videos uploaded to specific social networks (e.g., TikTok, WeiBo) present altered metadata, which invalidates metadata-based forensics methods. This is one of the limitations of our techniques and will be addressed in future research.

2.2 Related Work

Many techniques have been described to determine whether some part of a video has been tampered or not [3]. Most of these methods were developed to detect manipulations in the pixel domain and do not use metadata information. Compared to pixel-level analysis, metadata-based methods possess unique advantages. The size of metadata is significantly smaller than pixel data, which enables the analysis of large datasets in short amounts of time. Most video manipulation software do not allow users to alter metadata directly [14], [15]. Consequently, metadata has a higher degree of opacity than pixel data, which makes metadata-based media forensics more reliable and its corresponding attacks more challenging.

Most existing work focuses on the metadata in MP4-like video containers, which maintain data in a tree structure. In [15] and [17], the authors design features based on symbolic representation of

MP4's tree structure, which are processed by a statistical decision framework and decision trees, respectively. The authors report high performance for both video authentication and attribution tasks. Güera *et al.* [14] extract video metadata with the `ffprobe` utility and then do video authentication with an ensemble classifier.

More low-level metadata-related information can be found by looking into video compression traces. Video compression methods typically leave series of traces related to the way the video frames are compressed. This information is not easy to modify, thus acting as a metadata-like feature. As an example, most contemporary video encoders compress frame sequences in a structures known as a Group of Picture (GOP), where one frame can be defined using contents of other frames in order to save space [23]. The dependency between frames within or across different GOPs may provide evidence for video manipulation. Due to the complexity of video codecs, a number of techniques have been proposed for various settings of a codec where specific video encoding features are turned on or off. Vázquez-Padín *et al.* [24] provide a detailed explanation of the video encoding process and GOP structure. They propose a video authentication method that generalizes across multiple video encoding settings. Yao *et al.* [25] discuss the detection of double compression when an advanced video encoding feature called adaptive GOP is enabled.

2.3 Proposed Approach

2.3.1 An Overview of Our Approach

Video metadata captures multiple aspects of the history of a video file. In this chapter we propose a framework that exploits an MP4 video file's metadata to solve multiple video forensics problems, including brand attribution, video editing tool identification, social network attribution, and manipulation detection. Our method can also be easily adapted to other forensics applications.

The structure of our proposed framework is illustrated in Figure 2.1. As will be discussed in Section 2.3.2, the MP4 format manages data using a tree structure. First we extract the metadata from MP4 files while preserving their relationships in the tree structure. The MP4 standard is around twenty years old, it contains numerous vendor-specific nuances that require separate parsing strategies. The metadata tree needs to go through several refining stages, which increase the granularity of the extracted information. In the next step, the tree representation of metadata is

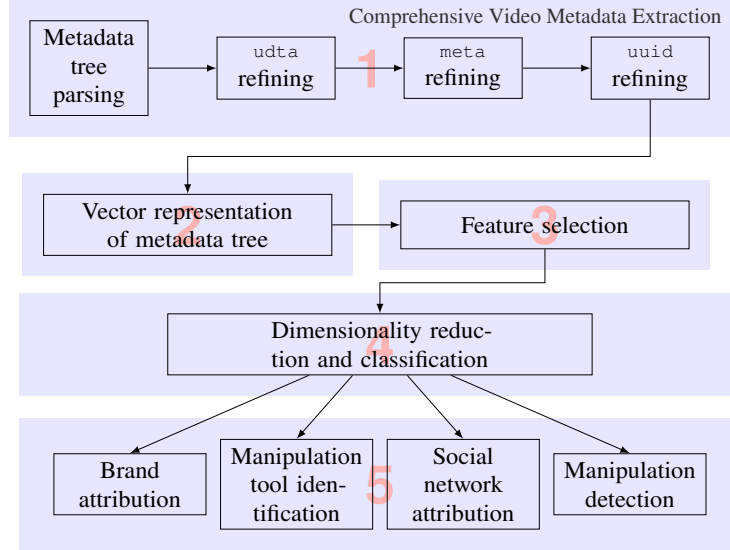


Figure 2.1. The structure of our proposed metadata forensic analysis technique.

converted into a numeric feature vector, which can be easily processed by machine learning methods. Our feature representation scheme is based upon [17]. We improve the handling of media tracks and metadata fields that take on continuous values inside the tree. The resulting feature vectors preserve more characteristics of the videos, yet they tend to also be more compact. In the last stage, we use these features with a classifier based on the selected forensic application. In the following we provide additional details about each step of our proposed framework.

2.3.2 Video Metadata

The first step in our approach consists of parsing metadata from video files. Digital video files are designed to handle multimodal data such as video, audio, and subtitles. The standards of these data modalities also shifts as technology advances. For example, since MP4’s introduction in 2001, the mainstream video codec has changed from MPEG-2 to H.264, and may change to H.265 in the near future [26]–[28]. The metadata surrounding these various data modalities and standards are inserted into a video file in distinct ways. In this chapter, we use the word *comprehensive* to describe a video metadata extraction scheme that is capable of parsing metadata across most data modalities and encoding specifications.

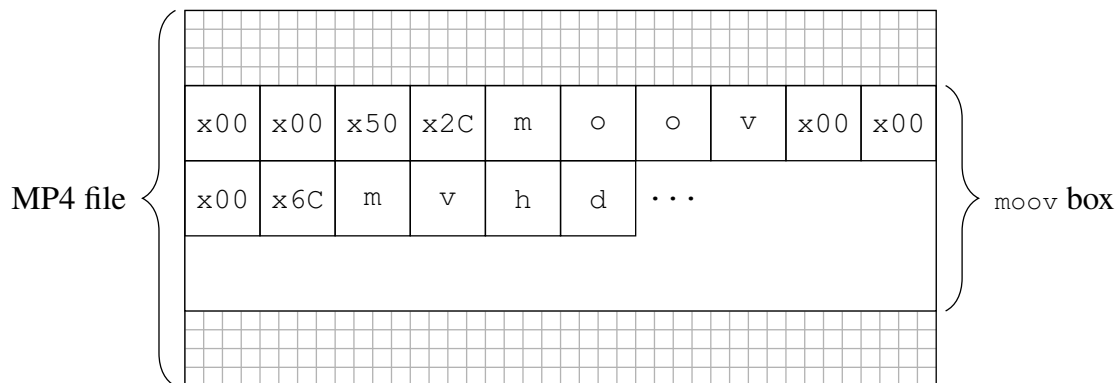


Figure 2.2. Illustration of the MP4 file format, where each cell represents one byte. An MP4 file is made up of a series of *boxes*. Every box has an 8-byte header, where the first 4 bytes store the size of the box in bytes (including the header) as a 32-bit big-endian integer, and the last 4 bytes store the name of the box. The content of a box is either child boxes or binary data. Binary data from different boxes may require distinct decoding methods.

Figure 2.2 shows the basic structure of an MP4 file [29]. An MP4 file is composed by a number of *boxes*. Each box starts with a header that specifies the box size (in byte) and name. The size helps the user to locate box boundaries, and the name defines the purpose of the box. The content of a box can either be a child box or some encoded binary data. Thus, an MP4 file is effectively a tree structure, where information is stored on leaf nodes. Given MP4's tree structure, we can capture metadata information at two levels: (1) the structure of the tree; and (2) the interpreted values of binary data contained in leaf nodes. Therefore, the job of a metadata extractor is to traverse the MP4 tree, attain the tree's structure, filter non-metadata leaf nodes (e.g., nodes that contain video compressed pixel information) and interpret values on relevant leaves. As shown in Figure 2.3, the output of a metadata extractor can be represented without any loss by a collection of human-readable strings.

Our metadata extractor focuses on vendor-specific non-standard MP4 metadata boxes that are significant for forensic purposes. More precisely, we determine that `udta`, `uuid`, `meta`, and `ilst` boxes are likely to carry vital information for video forensics. We next discuss our strategies to refine the parsing process.

STRING 1
moov
MEANING
The moov box is present in MP4 file.

STRING 2
moov/mvhd
MEANING
The mvhd box is present in MP4 file. It is a child box of moov. One can also imply that the moov node itself must not contain any binary data.

STRING 3
moov/mvhd/@duration=1546737
MEANING
The mvhd box is a leaf node in the tree; it contains a field called duration, whose value is 1546737.

STRING 4
moov/udta/@A9mod=0009&81iPhone 5c
MEANING
The udta box is a leaf node in the tree; it contains a field called A9mod, whose value is 0009&81iPhone 5c.

Figure 2.3. Examples of representing MP4 metadata tree with strings. Node paths are separated with ‘/’, the values of leaf nodes are prefixed with ‘@’, non-ASCII and unprintable characters are shown as hexadecimal codes surrounded by black frames. The metadata tree of any MP4 file can be portrayed by a collection of such strings.

2.3.2.1 Parsing `ilst` Data

`ilst` (“metadata item list”) boxes in MP4 files are used to store vendor-specific metadata [30]. Generally speaking, `ilst` boxes are container boxes whose child boxes carry metadata items as key-value pairs. The names of `ilst`’s children (i.e., the *keys*) would start with `A9` (equivalent to character ‘©’). A list of frequently used `ilst` keys is shown in Table 2.1. One can see that the content of the `ilst` box is particularly important for forensic analysis, for it often contains information about the manufacturer of the video capturing device, the encoder version, and the location and time of the capture.

It is difficult to parse the `ilst` box because various device manufacturers employ vastly different approaches when using it. Below, we report some interesting variants we found during our experiments:

- `ilst`'s child boxes directly placed in `moov/udta`

In some old Apple devices (e.g., iPhone 4, iPhone 5c, iPad 2), the child boxes of `ilst` are placed directly in `moov/udta` box.

- `ilst` boxes in `moov/meta`

As its name suggests, the `meta` box is used to store metadata. In this case, the `meta` box behaves similarly as other standard boxes, which means it can be parsed normally. As the MP4 parser traverses the box, it will eventually reach `ilst` and its children.

- `ilst` boxes in `moov/udta/meta` and `moov/trak/meta`

When `meta` boxes appear in `udta` and `trak` boxes, they deviate from standard boxes. More specifically, 4 extra bytes are inserted right after the `meta` header to store information [32]. These types of `meta` boxes cannot be parsed by the MP4 parser, normally because the program will see these 4 bytes as the size of next box, which will lead to corrupted results.

Our comprehensive metadata extractor is able to distinguish between these three scenarios and process MP4 video files correctly by fine-tuning the parsing of `udta` and `meta` boxes.

2.3.2.2 Parsing XML Data

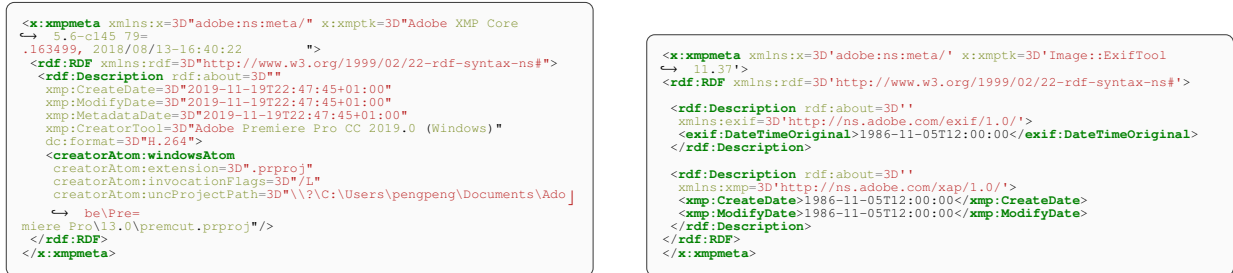
We concluded that many video files contain XML data after inspecting numerous video files, especially those edited by ExifTool and Adobe Premiere Pro. These tools make use of the Extensible

Table 2.1. A list of common keys in `ilst` boxes [31].

Key	Description
<code>A9mod</code>	camera model
<code>A9too</code>	name and version of encoding tool
<code>A9nam</code>	title of the content
<code>A9swr</code>	name and version number of creation software
<code>A9swf</code>	name of creator
<code>A9day</code>	timestamp of the video
<code>A9xyz</code>	geographic location of the video

Metadata Platform (XMP) to enhance metadata management, which introduces a large amount of metadata inside an MP4 file's `uuid` and `udta` boxes in the form of XML. In Figure 2.4, we show two XML examples extracted from MP4 containers. It can be seen that these XML data can potentially contain a large amount of information, which includes type of manipulation, original value before manipulation, and even traces to locate the person that applied the manipulation. It is vital for our metadata extractor to have the ability to handle XML data inside MP4 files.

To avoid over-complicating the extracted metadata tree, we discard the tree structure of XML elements and flatten them into a collection of key-value pairs. If there is a collision between key names, the older value will be overwritten by the newer one, which indicates that only the last occurring value of each key is preserved. Despite the fact that some information is lost by doing so, the data we have extracted is likely to be more than enough for most automated video forensic applications.



```

<x:xmpmeta xmlns:x=3D"adobe:ns:meta/" x:xmptk=3D"Adobe XMP Core
↳ 5.6-c145 79"
.163499, 2018/08/13-16:40:22 ">
<rdf:RDF xmlns:rdf=3D"http://www.w3.org/1999/02/22-rdf-syntax-ns#"
<rdf:Description rdf:about=3D""
xmp:CreateDate=3D"2019-11-19T22:47:45+01:00"
xmp:ModifyDate=3D"2019-11-19T22:47:45+01:00"
xmp:MetadataDate=3D"2019-11-19T22:47:45+01:00"
xmp:CreatorTool=3D"Adobe Premiere Pro CC 2019.0 (Windows)"
dc:format=3D"H.264">
<creatorAtom:windowsAtom
creatorAtom:extension=3D".prproj"
creatorAtom:invocationFlags=3D"/L"
creatorAtom:uncProjectPath=3D"\\?C:\Users\pengpeng\Documents\Ado
↳ be\Pre=
miere Pro\13.0\premcut.prproj"/>
</rdf:RDF>
</x:xmpmeta>

```

```

<x:xmpmeta xmlns:x=3D"adobe:ns:meta/" x:xmptk=3D"Image:ExifTool
↳ 11.37"
<rdf:RDF xmlns:rdf=3D"http://www.w3.org/1999/02/22-rdf-syntax-ns#"
<rdf:Description rdf:about=3D""
xmlns:exif=3D"http://ns.adobe.com/exif/1.0/"
<exif:DateTimeOriginal>1986-11-05T12:00:00</exif:DateTimeOriginal>
</rdf:Description>
<rdf:Description rdf:about=3D""
xmlns:xmp=3D"http://ns.adobe.com/xap/1.0/"
<xmp:CreateDate>1986-11-05T12:00:00</xmp:CreateDate>
<xmp:ModifyDate>1986-11-05T12:00:00</xmp:ModifyDate>
</rdf:Description>
</rdf:RDF>
</x:xmpmeta>

```

Figure 2.4. Examples of XML data in MP4 video containers.

2.3.3 Track and Type Aware Feature

The second step of our approach consists of turning the parsed metadata into feature vectors. Most machine learning methods use vectors as the input. The string representation of metadata trees generated in the previous step needs to be transformed into feature vectors before being used by machine learning methods.

Our feature representation technique is shown in Figure 2.5. For feature vectors to contain sufficient information of the MP4 tree, they need to include two levels of details: structure of the tree and value of metadata fields. Metadata can be either categorical or continuous numerical fields. Considering categorical values, we assign each node and metadata key-value pair in the MP4 tree an

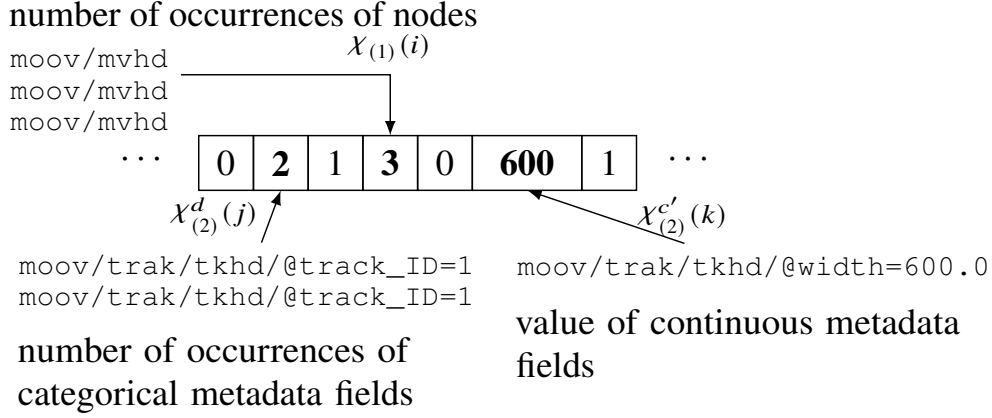


Figure 2.5. Illustration of the vector representation of MP4 metadata. The χ functions help determine the corresponding element of a node or a metadata field in the feature vector.

element in the feature vector, which counts the number of occurrences of that node or pair. This strategy preserves information about the MP4 tree in the feature vector to a great extent. Considering numerical values, creating a new element for each of these key-value pairs will render the feature vectors large and redundant. We decide to insert the values into the feature vectors directly.

From Figure 2.3, we know that string representation can be put into two categories: (1) strings that indicate the presence of a node, with node path separated by '/'; and (2) strings that show the key-value pair stored in a node, with node path separated by '/' and key-value separated by '='. Since an MP4 file can be seen as a collection of such strings, the feature transformation process can be viewed as a mapping from a given collection of strings S to a vector v . For the discussion below, we use $v_{[l]}$ to denote the l -th element of v .

In order to construct a mapping $S \rightarrow v$, we need to consider the set of all possible strings Ω . We denote the set of all category (1) strings and category (2) strings by $C_{(1)}$ and $C_{(2)}$, respectively. By definition, $C_{(1)}$ and $C_{(2)}$ form a partition of Ω . We assume that both $C_{(1)}$ and $C_{(2)}$ are ordered so that we can query each element by its index. Let us denote the l -th elements of $C_{(1)}$ and $C_{(2)}$ by $C_{(1)}[l]$ and $C_{(2)}[l]$, respectively.

Each category (1) string corresponds to an element in v . For the i -th category (1) string, we denote the index of the corresponding vector element in v by $\chi_{(1)}(i)$. The value corresponding to this element is given by

$$v[\chi_{(1)}(i)] = \text{number of times } C_{(1)}[i] \text{ occurs in } S, \\ \forall i \in \{1, 2, \dots, |C_{(1)}|\}. \quad (2.1)$$

We treat each media track (`trak`) segment in the metadata strings in a different way. In the MP4 file format, each `trak` node contains information about an individual track managed by the file container. We observed that the number of tracks and content of the tracks remain consistent among devices of the same model. The structure of an MP4 file can be better preserved if we distinguish different tracks rather than merging them. This is achieved by assigning each track a track number in the metadata extraction phase. For example, the first track will be `moov/trak1`, and the second track will be `moov/trak2`. As a result, the child nodes and key-value pairs of each track will be separated, which effectively makes the feature vectors *track-aware*.

For category (2) strings that represent key-value pairs stored in a node, we applied a slightly different transformation strategy. We observed that some fields in MP4 files are essentially *continuous* (e.g., `@avgBitrate`, `@width`). Despite the fact that most MP4 metadata fields are discrete, assigning each combination of these continuous key-value pairs a new element in v will still result in large and redundant feature vectors. We continue to subdivide $C_{(2)}$ based on the *type* of each field, where the set of strings that have discrete fields is denoted by $C_{(2)}^d$, and the set of strings that have continuous fields is denoted by $C_{(2)}^c$. For strings that belong to $C_{(2)}^d$, the transformation scheme is similar to that of category (1) strings. Let the vector element index of the j -th string in $C_{(2)}^d$ be $\chi_{(2)}^d(j)$, then the value corresponding to the element is given by

$$v[\chi_{(2)}^d(j)] = \text{number of times } C_{(2)}^d[j] \text{ occurs in } S, \\ \forall j \in \{1, 2, \dots, |C_{(2)}^d|\}. \quad (2.2)$$

As for strings that belong to $C_{(2)}^c$, we first discard the values in the strings to form a set of distinct keys $C_{(2)}^{c'}$, and then put the values in \mathbf{v} directly. For the k -th string in $C_{(2)}^{c'}$, the index of the corresponding vector element in \mathbf{v} is $\chi_{(2)}^{c'}(j)$, and the value of the element is

$$\mathbf{v}[\chi_{(2)}^{c'}(k)] = \begin{cases} \text{the value of key } C_{(2)}^{c'}[k] \text{ in } S \\ 0 \text{ (if the key } C_{(2)}^{c'}[k] \text{ is not in } S) \end{cases}, \quad \forall j \in \{1, 2, \dots, |C_{(2)}^{c'}|\}. \quad (2.3)$$

It can be seen that the dimensionality of \mathbf{v} is given by

$$\dim(\mathbf{v}) = |C_{(1)}| + |C_{(2)}^d| + |C_{(2)}^{c'}|. \quad (2.4)$$

In general, the actual value of the index functions χ can be arbitrary as long as all values of $\chi_{(1)}(i)$, $\chi_{(2)}^d(j)$, and $\chi_{(2)}^{c'}(k)$ form a valid partition of integers in the range $[1, \dim(\mathbf{v})]$.

By using different representation strategies for discrete and continuous fields (i.e., being *type-aware*), the resulting feature vectors are more compact and suited to machine learning techniques.

2.3.4 Feature Selection

Our third step consists of reducing the set of selected features by discarding redundant features. Based on the feature extraction scheme above, it can be observed that some elements in the feature vector are significantly correlated. For example, the presence of string `moov/mvhd/@duration=1546737` in a collection S extracted from a valid MP4 file must lead to the presence of `moov/mvhd` in S . Therefore, feature selection can reduce redundancy within the feature vectors.

In the feature selection step, we reduce the redundancy among features that are strongly correlated. Since only a small proportion of elements in the feature vector \mathbf{v} are inserted from continuous fields, most elements in \mathbf{v} correspond to the number of occurrences of a node or a field value. If two features in \mathbf{v} are negatively correlated, then it often means the presence of an MP4 entity implies the absence of another MP4 entity. In forensic scenarios, presence is much stronger evidence than

absence. Therefore, if we only focus on features that are positively correlated, then we can select features of higher forensic significance.

Given a set of feature vectors v_1, v_2, \dots, v_N , we can compute the corresponding correlation matrix \mathbf{R} , where R_{ij} is equal to the correlation coefficient between i -th and j -th feature in the set. Then, we set all negative elements in \mathbf{R} to zero, which results in matrix \mathbf{R}^+ . That is, negative correlation is ignored. Because all elements in \mathbf{R}^+ are within the range $[0, 1]$, the matrix \mathbf{R}^+ can be seen as an affinity matrix between $\dim(v)$ vertices in a graph, where an affinity value of 0 indicates no connection and an affinity value of 1 indicates strongest connection. This allow us to use spectral clustering with α clusters on \mathbf{R}^+ , which assigns multiple strongly correlated features into the same cluster [33]. See Section 2.6.1 for more details about spectral clustering. Then, we select clusters with more than β features. For each selected cluster, we retain only one feature at random. Here, $\alpha > \beta > 0$ are hyperparameters. This feature selection step helps improve the quality of feature vectors.

2.3.5 Dimensionality Reduction and Classification

In the last step, depending on the the video forensics problem, we use the feature vectors for classification in two ways.

2.3.5.1 Multi-class problems

When the classification problem is multi-class, we use Linear Discriminant Analysis (LDA) [34] to drastically reduce the dimensionality of feature vectors to 2 dimensions. LDA is a supervised dimensionality reduction technique. For more details about LDA, see Section 2.6.2. For a classification problem of K classes, LDA generates an optimal feature representation in a subspace that has dimensionality of at most $K - 1$. The new features in the subspace are optimal in the sense that the variance between projected class centroids are maximized. For multi-class classification problems, we always reduce the dimensionality of the feature vector to 2. After dimensionality reduction, we use a nearest neighbor classifier that uses the distance between the query sample and λ nearest samples to make a decision, where λ is a hyperparameter. Each nearest sample is weighted by their distance to the query sample. The use of the dimensionality reduction and nearest neighbor

classifier lead to concise and straightforward decision rules in 2D space, which can be interpreted and analyzed by human experts easily.

2.3.5.2 Two-class problems

When the classification problem is two-class ($K = 2$), LDA can only generate one-dimensional features. Our experiments have shown that 1D features are insufficient to represent the underlying complexity of video forensics problems. As a result, for binary classification problems, we use a decision tree classifier without dimensionality reduction.

2.4 Experiments and Results

In this section we report all the details of the experiments and comment on the results.

We study the effectiveness of our approach using the following datasets:

- VISION [35]: the VISION dataset contains 629 pristine MP4/MOV videos captured by 35 different Apple, Android and Microsoft mobile devices.
- EVA-7K [17]: the EVA-7K dataset contains approximately 7000 videos captured by various mobile devices, uploaded to distinct social networks, and edited by different video manipulation tools. The authors took a subset of videos from the VISION dataset and edited them with a number of video editing tools. Then, they uploaded both the original videos and edited videos to four social networks, namely YouTube, Facebook, TikTok and WeiBo. The videos were subsequently downloaded back. The EVA-7K dataset is made up of the pristine videos, edited videos, and downloaded videos.

The VISION dataset is used for device attribution experiments, while all other experiments are conducted on the larger EVA-7K dataset.

We demonstrate the effectiveness of our approach in four video forensic scenarios. In all of the experiments below that use LDA and nearest neighbor classification, we choose $\alpha = 300$, $\beta = 4$, and $\lambda = 5$. For each of the scenarios below, unless indicated otherwise, the dataset is split into two halves with stratified sampling for training and validation, respectively. The metadata nodes and fields that are excluded during metadata extraction, as well as the list of continuous features in

the vector representation step, are provided in supplementary materials. We mainly compare the performance of our method to [17], where the EVA-7K dataset is described. For brand attribution, because it is conducted on the VISION dataset, we select [15] as the baseline. The experiment results show that our approach achieves high performance evaluation metrics in all four scenarios.

2.4.1 Scenario 1: Brand Attribution

Brand attribution consists of identifying the brand of the device used to capture the video file under analysis. We examined brand attribution experiments in two scenarios. In the first experiment, we assume the analyst has access to all possible devices at training time (i.e., close-set scenario). In the second experiment, we assume that a specific device may be unknown at training time (i.e., blind scenario).

2.4.1.1 Close-set scenario

In Table 2.2, we show the F1-score comparison between our approach and previous work. Our method almost perfectly classifies the VISION dataset, with only one Apple device being misclassified as Samsung. Because our framework is capable of extracting and analyzing more metadata, the performance of our method is better compared to the baseline, especially for brands like Huawei, LG, and Xiaomi. The 2D feature distribution and decision boundary for each label are shown in Figure 2.6, from which we can determine the metadata similarity between brands and the metadata “variance” for each brand. Visualizations as shown in Figure 6 generated by our method can aid an analyst in making a more interpretable and reliable decision. If a new video under examination is projected close to the LG region, one can assume it is unlikely for that video to come from a Samsung device.

2.4.1.2 Blind scenario

We also examined device attribution with an unknown device using our approach. Let us consider a specific example where device ID 20 (an Apple device) is the unknown device. In the training phase, we use the entire VISION dataset except for samples from device ID 20. In order to classify this device that is unknown to the classifier, we project its features in 2D space and plot

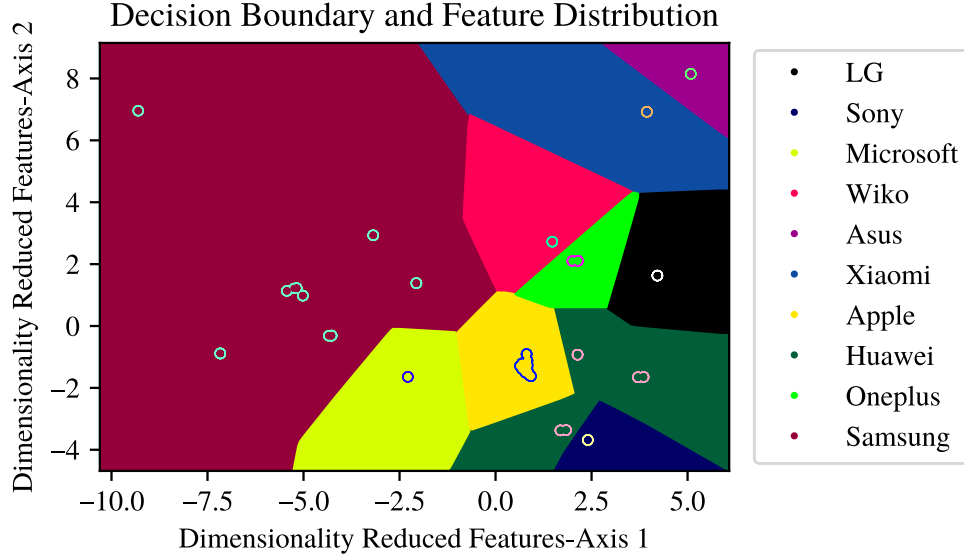


Figure 2.6. 2D feature distribution and classification boundary for device attribution scenario.

Table 2.2. F1-score comparison of device attribution scenario.

Brand	Device Attribution	
	Iuliani, <i>et al.</i> [15]	Our Approach
Apple	1.00	0.99
Asus	1.00	1.00
Huawei	0.87	1.00
LG	0.94	1.00
Oneplus	0.93	1.00
Samsung	0.93	0.99
Wiko	0.65	1.00
Xiaomi	0.74	1.00

them in the decision boundary plot shown in Figure 2.7. It can be seen that all samples lie in Apple’s decision region, which indicates that the unknown samples are classified correctly.

2.4.2 Scenario 2: Manipulation Tool Identification

The goal of this scenario is to determine the video editing tool used to manipulate a given video file. We considered native video files from the acquisition devices as non-edited, and compare them

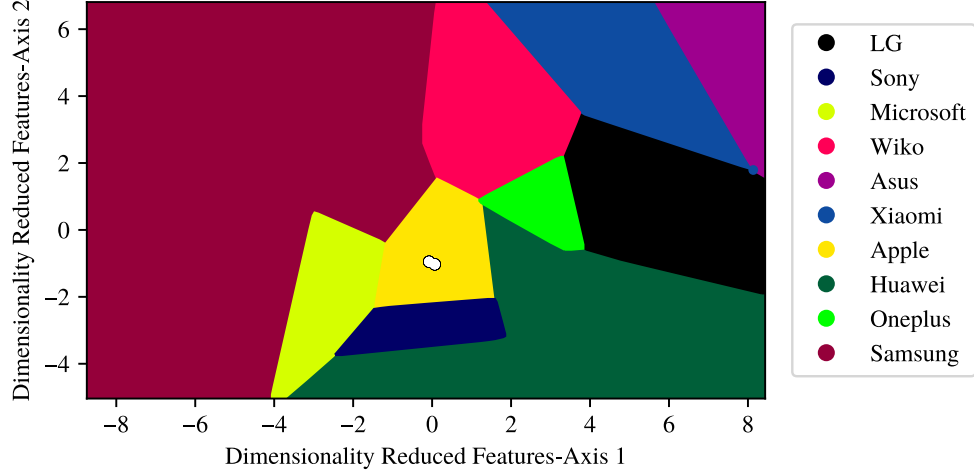


Figure 2.7. Example of blind device attribution scenario. The projected samples from the unknown device are shown with white markers with black contour.

with video files edited using Avidemux [36], Exiftool [37], ffmpeg [38], Kdenlive [39] and Premiere [40] as reported in [17].

In Table 2.3, we compare the F1-score of our method to previous work. Our method achieves a higher average F1-score compared to the previous work. The 2D feature distribution and decision boundary for this scenario are shown in Figure 2.8.

Table 2.3. F1-score comparison of manipulation tool identification scenario.
Manipulation Tool Identification

Tool	Yang, <i>et al.</i> [17]	Our Approach
Native	0.97	1.00
Avidemux	0.99	0.98
Exiftool	0.98	1.00
ffmpeg	0.94	1.00
Kdenlive	0.95	1.00
Premiere	1.00	0.99
Average	0.97	0.99

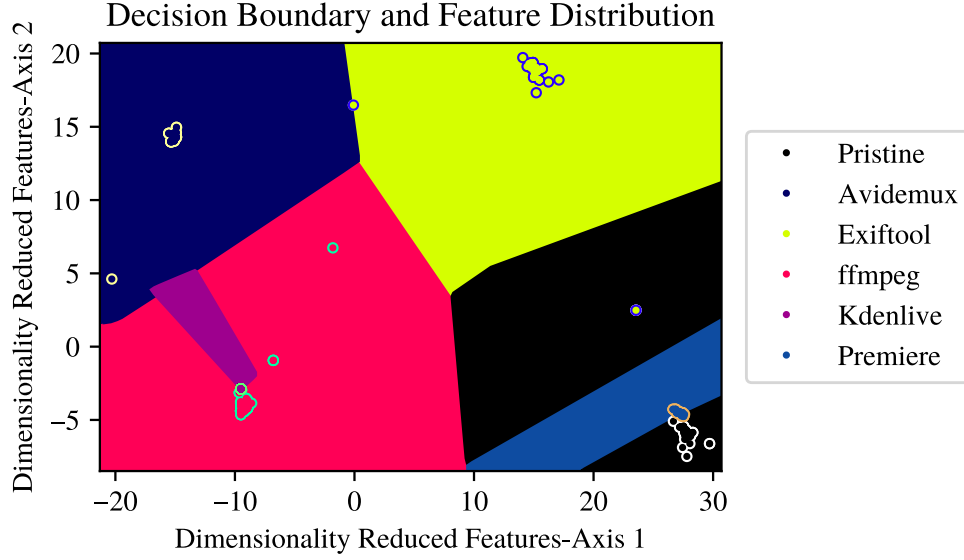


Figure 2.8. 2D feature distribution and classification boundary for manipulation tool identification scenario.

2.4.3 Scenario 3: Social Network Attribution

In our social network attribution scenario, we classify the source social network of the video files. If a video file is not downloaded from a social network or the video file comes from an unknown social network, it will be classified as “other”. The F1-scores in this scenario are shown in Table 2.4; the 2D feature distribution and decision boundary for this scenario are shown in Figure 2.9. For this task, our approach achieves high average F1-score of 0.99. The high performance also implies that each social network leaves a unique fingerprint on its videos.

Table 2.4. F1-score comparison of social network attribution scenario.

Social Network Attribution		
Social Network	Yang, <i>et al.</i> [17]	Our Approach
YouTube	1.00	0.99
Facebook	1.00	1.00
WeiBo	0.99	0.99
TikTok	1.00	1.00
Other	-	0.99

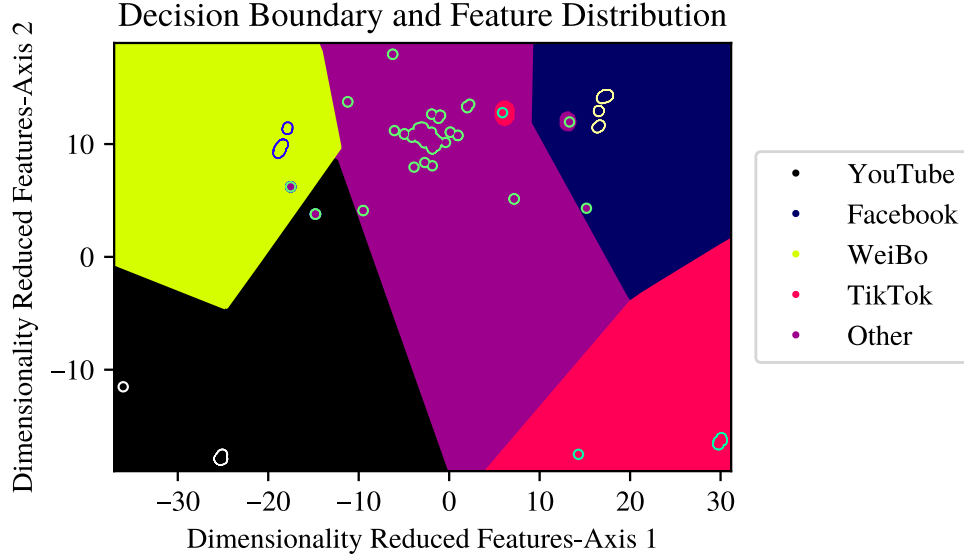


Figure 2.9. 2D feature distribution and classification boundary for social network attribution scenario.

2.4.4 Scenario 4: Manipulation Detection

There are two sub-tasks within this scenario. As discussed above, for binary classification scenarios such as manipulation detection, LDA can only generate one-dimensional features. It limits the features' expression power after dimensionality reduction, which leads to inferior classification performance. Therefore, for binary classification problems, we prefer using a decision tree classifier without dimensionality reduction.

2.4.4.1 Manipulation detection on videos files from social networks

In this task, we detect manipulated videos given the fact that both pristine and edited videos are uploaded to social networks first. We compare the performance of using our features with different classification strategies to [17], and the results are shown in Table 2.5. Using the EVA-7K dataset, the manipulation detection problem is unbalanced because there are much more edited videos than original ones (edited:pristine \approx 9:1), meaning more samples with positive labels. In this case, the true positive rate (TPR) and the true negative rate (TNR) reflect a classifier's performance more objectively than F1-score. It can be seen that our features combined with decision tree classifier

achieve higher TNR for videos from all four social networks. When we use LDA and nearest neighbor classifier for this scenario, the classifier completely fails for videos from TikTok and WeiBo. It is likely because LDA can only generate one-dimensional features, which do not possess enough degrees of freedom to represent the complexity of this problem. Thus, the decision tree classifier is preferred for this scenario.

From Table 2.5, we also have a glimpse of how each social network process uploaded videos. For WeiBo and TikTok videos, conducting further metadata-based forensic analysis becomes unreliable, which indicates they may have significantly altered videos uploaded to their platforms. YouTube videos can be classified perfectly, which implies that they apply minimum modification to videos' metadata.

Table 2.5. Performance evaluation metrics comparison between our approach and previous work. TPR and TNR stand for True Positive Rate and True Negative Rate, respectively. The accuracy score has been balanced.

Social Network Manipulation Detection				
		Yang, <i>et al.</i> [17]	Our Features+ Decision Tree	Our Features+ LDA & NNC
Facebook	Accuracy	0.76	0.84	0.62
	TNR	0.40	0.87	0.30
	TPR	0.86	0.82	0.95
TikTok	Accuracy	0.80	0.69	0.50
	TNR	0.51	0.94	0.00
	TPR	0.75	0.43	1.00
WeiBo	Accuracy	0.79	0.63	0.50
	TNR	0.45	0.57	0.00
	TPR	0.82	0.68	1.00
YouTube	Accuracy	0.60	1.00	1.00
	TNR	0.36	1.00	1.00
	TPR	0.74	1.00	1.00

2.4.4.2 Manipulation detection on local videos

In this task, we classify pristine videos and edited videos that are not exchanged via social network. To mimic the real world classification scenario, we employ a similar leave-one-out

validation strategy as introduced in [17]. This approach takes the video files from one device model out as the validation set at a time. Since there is only one Microsoft device among 35 models, it is discarded in this scenario, as described in [17]. Because the Microsoft device either belongs to the training set or validation set, we are left with no samples to validate or no data to train. The mean balanced accuracy comparison of the 34-fold cross validation is shown in Table 2.6. Our approach achieves higher performance compared to previous works.

Table 2.6. Comparison of our method with previous works. The balanced accuracy is averaged over 34 folds.

Manipulation Detection on Local Videos	
	Balanced Accuracy
Güera, <i>et al.</i> [14]	0.67
Iuliani, <i>et al.</i> [15]	0.85
Yang, <i>et al.</i> [17]	0.98
Our Features + Decision Tree	0.99

2.5 Conclusion

In this chapter, we proposed a video forensics approach for MP4 video files based on metadata embedded in video files. Our improved metadata extraction and feature representation scheme allows one to represent more metadata information in a compact feature vector. We use feature selection, dimensionality reduction, and nearest neighbor classification techniques to form interpretable and reliable decision rules for multiple video forensics scenarios. Our approach achieves better performance than other methods.

The performance of our method in many of the scenarios indicates that we need to increase our video forensics dataset to include more difficult cases. Our research also exposed the limitation of metadata-based forensics methods, namely its failure to analyze videos from specific social networks such as TikTok and WeiBo. This is a significant disadvantage compared to pixel-based methods. In the future, we plan to continue exploring the potential of metadata-based video forensics by adding the ability to parse more manufacturer-specific data models (e.g., Canon’s CNTH tags [41]) and by looking into lower-level metadata in the distribution of audio/video samples as well as location

of key frames in the video stream. We hope that metadata-based video forensics methods can be proved to be reliable in more forensic scenarios.

2.6 Supplementary Materials

The notations used in this section are independent from those used in other sections.

2.6.1 Spectral Clustering

Spectral clustering [33], [42], [43] is an approach to cluster data points given the pairwise similarity information between any two given data points. Suppose there are K data points $\mathbf{x}_1, \dots, \mathbf{x}_K$. The pairwise similarity information can be stored in a $K \times K$ symmetric matrix \mathbf{A} (known as the affinity matrix), where

$$\mathbf{A}_{ij} = \text{sim}(\mathbf{x}_i, \mathbf{x}_j). \quad (2.5)$$

Note that $\mathbf{A}_{ij} \in [0, 1]$. Define the degree matrix as

$$\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}. \quad (2.6)$$

First, consider clustering the data points into two clusters. As shown in [33], the spectral clustering problem can be formulated as the normalized cut of a weighted graph. The solution to this problem is a vector $\mathbf{y} \in \{-1, 1\}^K$ that assigns each data point to one of the two clusters. Let $\mathbf{z} = (3 - \mathbf{y})$. The optimal solution to the normalized cut problem can be acquired by solving the following optimization problem [33]:

$$\begin{aligned} \min_{\mathbf{y}} \quad & \frac{\mathbf{z}^T (\mathbf{D} - \mathbf{W}) \mathbf{z}}{\mathbf{z}^T \mathbf{D} \mathbf{z}}, \\ \text{s.t. } & \mathbf{y}_i \in \{-1, 1\}, \mathbf{z}^T \mathbf{D} \mathbf{1} = 0. \end{aligned} \quad (2.7)$$

The optimization problem in Equation (2.7) is NP hard. An approximate solution can be acquired by relaxing the constraint on \mathbf{y} such that it can take real values. With this relaxation, let $\mathbf{w} = \mathbf{D}^{-\frac{1}{2}}\mathbf{z}$, and the optimization problem is equivalent to solving the following linear system:

$$\mathbf{D}^{-\frac{1}{2}}(\mathbf{D} - \mathbf{A})\mathbf{D}^{-\frac{1}{2}}\mathbf{w} = \lambda\mathbf{w}. \quad (2.8)$$

Let

$$\mathbf{L} = \mathbf{D}^{-\frac{1}{2}}(\mathbf{D} - \mathbf{A})\mathbf{D}^{-\frac{1}{2}}, \quad (2.9)$$

which is also known as the Graph Laplacian (GL) in spectral graph theory [44]. The solution to \mathbf{w} corresponds to the eigenvector of the second smallest eigenvalue of \mathbf{L} [33].

The eigenspace based approach can be extended to clustering the data points into more than two clusters. Ng *et al.*[43] showed that the eigenvectors of \mathbf{L} can be used to project the original data points onto clusters with mutually orthogonal centroids on a unit sphere. Therefore, an algorithm to cluster the data points is described as follows. Suppose the data points are clustered into C clusters ($2 \leq C < K$).

1. Find $\mathbf{w}_1, \dots, \mathbf{w}_C$, the eigenvectors correspond to the C -smallest eigenvalues of \mathbf{L}
2. Build data matrix

$$\mathbf{X}' = [\mathbf{w}_1 | \dots | \mathbf{w}_C] \in \mathbb{R}^{K \times C}, \quad (2.10)$$

where the i -th row of \mathbf{X}' is \mathbf{x}_i in the projected space

3. Normalize each row of \mathbf{X}' to form $\tilde{\mathbf{X}}'$ so that each row in $\tilde{\mathbf{X}}'$ has unit length
4. Use any clustering algorithm (e.g., k -means clustering [45]) to cluster the normalized data points in the projected space (i.e., each row of $\tilde{\mathbf{X}}'$); the cluster labels assigned for the projected data points are used for the original data points

2.6.2 Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis (LDA) [46], [47] is a supervised dimensionality reduction approach. Suppose there are K data points with C different labels $(\mathbf{x}_i, y_i)_{i=1}^K$, where $\mathbf{x}_i \in \mathbb{R}^M$ and $y_i \in \{1, \dots, C\}$. LDA aims to find a linear projection $\mathbf{W}^T \in \mathbb{R}^{N \times M}$ that maps the data points to an N -dimensional subspace, while maximizing between class variance and minimizing within class variance. Let \mathcal{I}_i be the index set of the i -th class. Define the class mean as

$$\boldsymbol{\mu}_i = \frac{1}{|\mathcal{I}_i|} \sum_{j \in \mathcal{I}_i} \mathbf{x}_j, \quad (2.11)$$

and the overall mean as

$$\boldsymbol{\mu} = \sum_{i=1}^C \frac{|\mathcal{I}_i|}{K} \boldsymbol{\mu}_i. \quad (2.12)$$

The between class covariance matrix is given by

$$\mathbf{S}_b = \sum_{i=1}^C |\mathcal{I}_i| (\boldsymbol{\mu}_i - \boldsymbol{\mu})(\boldsymbol{\mu}_i - \boldsymbol{\mu})^T. \quad (2.13)$$

The within class covariance matrix is given by

$$\mathbf{S}_w = \sum_{i=1}^C \sum_{j \in \mathcal{I}_i} (\mathbf{x}_j - \boldsymbol{\mu}_i)(\mathbf{x}_j - \boldsymbol{\mu}_i)^T. \quad (2.14)$$

After the applying the projection \mathbf{W}^T , the between class covariance and within class covariance are $\mathbf{S}'_b = \mathbf{W}^T \mathbf{S}_b \mathbf{W}$ and $\mathbf{S}'_w = \mathbf{W}^T \mathbf{S}_w \mathbf{W}$, respectively.

To maximize between class variance and minimize within class variance after projection, the following function is used as optimization objective [47], [48]:

$$f(\mathbf{W}^T) = \text{tr}(\mathbf{S}'_w^{-1} \mathbf{S}'_b). \quad (2.15)$$

Using matrix calculus , it can be seen that

$$\frac{\partial f(\mathbf{W})}{\partial \mathbf{W}} = -2\mathbf{S}_w \mathbf{W} (\mathbf{W}^T \mathbf{S}_w \mathbf{W})^{-1} \mathbf{W}^T \mathbf{S}_b \mathbf{W} (\mathbf{W}^T \mathbf{S}_w \mathbf{W})^{-1} + 2\mathbf{S}_b \mathbf{W} (\mathbf{W}^T \mathbf{S}_w \mathbf{W})^{-1} \quad (2.16)$$

$$= -2\mathbf{S}_w \mathbf{W} \mathbf{S}_w'^{-1} \mathbf{S}_b' \mathbf{S}_w'^{-1} + 2\mathbf{S}_b \mathbf{W} \mathbf{S}_w'^{-1}. \quad (2.17)$$

Setting the derivative to zero gives

$$(\mathbf{S}_w^{-1} \mathbf{S}_b) \mathbf{W} = \mathbf{W} (\mathbf{S}_w'^{-1} \mathbf{S}_b'). \quad (2.18)$$

Since \mathbf{S}_b' and \mathbf{S}_w' are both symmetric, they can be simultaneously diagonalized using an invertible matrix \mathbf{B} [48] so that

$$\mathbf{B}^T \mathbf{S}_b' \mathbf{B} = \mathbf{\Lambda}_b' \quad (2.19)$$

$$\mathbf{B}^T \mathbf{S}_w' \mathbf{B} = \mathbf{I}, \quad (2.20)$$

where $\mathbf{\Lambda}_b'$ is a diagonal matrix and \mathbf{I} is the identity matrix. From the properties of \mathbf{B} , it can be derived that

$$\mathbf{S}_w'^{-1} \mathbf{S}_b' \mathbf{B} = \mathbf{B} \mathbf{\Lambda}_b'. \quad (2.21)$$

That is, the columns of \mathbf{B} are the eigenvectors of $\mathbf{S}_w'^{-1} \mathbf{S}_b'$, and the diagonal elements of $\mathbf{\Lambda}_b'$ are the eigenvalues of $\mathbf{S}_w'^{-1} \mathbf{S}_b'$. The properties of \mathbf{B} also allow us to rewrite Equation (2.18) as

$$(\mathbf{S}_w^{-1} \mathbf{S}_b) (\mathbf{W} \mathbf{B}) = (\mathbf{W} \mathbf{B}) \mathbf{\Lambda}_b', \quad (2.22)$$

which shows that $\mathbf{\Lambda}_b'$ is also the eigenvalues of $\mathbf{S}_w^{-1} \mathbf{S}_b$. Because the value of trace is equal to the sum of all eigenvalues, we can maximize $\text{tr}(\mathbf{S}_w'^{-1} \mathbf{S}_b')$ by selecting the top eigenvalues of $\mathbf{S}_w'^{-1} \mathbf{S}_b'$. The matrix \mathbf{W}^T is formed by the corresponding eigenvectors of $\mathbf{S}_w^{-1} \mathbf{S}_b$.

3. FORENSIC ANALYSIS AND LOCALIZATION OF MULTIPLY COMPRESSED MP3 AUDIO USING TRANSFORMERS

3.1 Introduction

The advance in machine learning techniques makes generating high-quality speech or music possible [49]–[51]. Almost everyone has the ability to tamper with audio signals due to the availability of audio editing techniques. It is easy to synthesize/manipulate “fake” speech signals that resembles the style and voice of a given person, and to concatenate real and synthetic signals to create new forged speech signals. This poses significant threats to individuals, organizations, society and national security.

The detection of synthesized/manipulated speech is usually challenging because of the flexibility of human voice, the presence of acoustic noise or reverberation, and the complexity of audio synthesis models [52]. However, audio signals are mostly saved and shared using lossy compression techniques. Lossy compression leaves distinct artifacts in the compressed audio which can be used for forensic analysis [53], [54]. In this chapter we examine the forensic problem of detecting if an audio signal has been spliced into another audio signal by detecting locations in the signal that have been multiply compressed. The spliced audio signal may be real or synthetic.

Introduced in 1993, MPEG-1 Audio Layer III (MP3) [55], [56] has been one of the most popular digital audio compression methods. It changed our ways of listening to music, podcasts, and many other types of audio content. Despite having inferior compression efficiency compared to Advanced Audio Codec (AAC) [57], MP3 is still widely used due to compatibility with many existing applications and lower computational complexity [58].

Most of the existing MP3 audio forensics methods focus on double MP3 compression detection. These methods predict whether an entire MP3 file is compressed more than once. In [53], [54], [59], [60], the authors used different statistical and feature design techniques on the Modified Discrete Cosine Transform (MDCT) coefficients for double compression detection. Ma, Wang, Yan, and Jin [61] used the statistics of scalefactors for detecting doubly compressed MP3 with the same data rate. In [62], the authors used the Huffman table indices for double compression detection. Yan, Wang, Zhou, Jin, and Wang [63] addressed the problem of double and triple compression detection using features extracted from scalefactors and Huffman table indices. In [64], the authors addressed the

MP3 multiple compression localization problem for the first time. Essentially, their proposed method detects double compression using the histogram of MDCT coefficients. Localization was achieved by using small sliding detection windows. This technique allows one to extend any detection method to a localization method. Finally, more modern methods make use of Deep Neural Networks (DNNs). This is the case of Luo, Cheng, Yuan, Luo, and Liu [65] that proposed to use stacked autoencoder networks to detect multiply compressed audio signals.

In this chapter, we present an MP3 multiple compression localization technique based on deep transformer neural networks [66]. Given an MP3 signal, our proposed method can distinguish between single compressed temporal segments and multiple compressed temporal segments thereby allowing us to temporally localize where the audio signal may have been spliced. Our proposed method can also be used for synthesized audio detection.

3.2 Background

We first introduce a basic overview of MP3 compression. More details of MP3 compression can be obtained in [55], [56], [58]. Then we provide an overview of transformer neural networks.

3.2.1 MP3 Compression

The block diagram of a typical MP3 encoder is shown in Figure 3.1. To compress a digital audio signal using MP3, the signal is first split into fixed time length sample windows known as frames, where each frame contains 1152 temporal samples [55], [56]. MP3 files are made up of a series of such frames.

The input is first processed through a perceptual model that drives the selection of coding parameters (lower branch of Figure 3.1). During this step, the audio samples are transformed into frequency domain using Fast Fourier Transform (FFT). The FFT magnitude is passed to the psychoacoustic model, which exploits the characteristics of Human Hearing System (HHS) to balance between the sound quality and the data rate of the compressed signal [67]. After this perceptual audio analysis, the lossy coding step is next (upper branch of Figure 3.1). The temporal samples are filtered into 32 equally spaced frequency sub-bands using a polyphase filterbank. Each sub-band is windowed according to the psychoacoustic model to reduce artifacts and then

transformed through MDCT, which leads to 18 coefficients. In total, there are $32 \times 18 = 576$ MDCT coefficients.

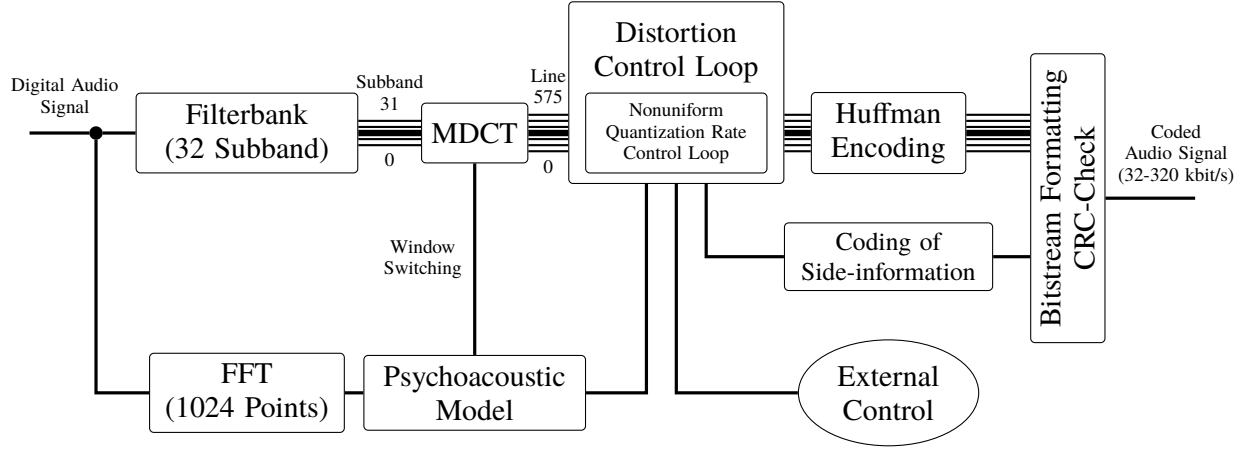


Figure 3.1. The block diagram of an MP3 encoder.

After the MDCT, the resulting coefficients are quantized in the distortion control loop exploiting the psychoacoustic analysis. The HHS has approximately 24 critical bands based on a model of the HHS [67]. During quantization, the 32 sub-bands are grouped into scalefactor bands, which approximates the critical bands of HHS. The MDCT coefficients in each scalefactor band is multiplied by a scalefactor before quantization. The quantization step size increases as the frequencies become greater, because the HHS is less sensitive to higher frequency. The scalefactors and quantization step sizes work together to satisfy both audio quality and data rate constraints. After quantization, the MDCT coefficients are binary encoded using one of the 32 predefined Huffman tables. The binary coded coefficients, the encoding parameters (side-information) such as scalefactors, Huffman table indices, quantization step sizes are inserted in the data stream to form the compressed audio signal.

3.2.2 Transformer Neural Networks

Transformer networks have shown excellent performance in a variety of tasks such as language modeling [68], image classification [69], object detection [70], protein structure prediction [71].

Let the input to the transformer network be $\mathbf{Z} \in \mathbb{R}^{N \times d_{\text{model}}}$, which contains N elements, and each element is a vector of d_{model} dimensions. Transformer networks use the Self Attention (SA) mechanism [66], [69] to exploit the relationship between different elements in the input. Linear

projection $U_{qkv} \in \mathbb{R}^{d_{\text{model}} \times 3d_h}$ is first used to generate three different projected versions of the input, i.e., q , k , and v :

$$[q \mid k \mid v] = ZU_{qkv}. \quad (3.1)$$

Then, the vectors q and k are used to form the attention map $A \in \mathbb{R}^{N \times N}$:

$$A = \text{softmax} \left(qk^T / \sqrt{d_h} \right). \quad (3.2)$$

Finally, the SA of Z is the matrix multiplication of A and v :

$$\text{SA}(Z) = Av. \quad (3.3)$$

The transformer networks we use are based on Multihead Self Attention (MSA) [66], which is an extension of SA where h different SA values (called “heads”) are computed in parallel. The h different SA values are combined to the result of MSA using the matrix $U_{msa} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$:

$$\text{MSA}(Z) = [\text{SA}_1(Z) \mid \cdots \mid \text{SA}_h(Z)] U_{msa}. \quad (3.4)$$

One must guarantee h divides d_{model} and set $d_h = d_{\text{model}}/h$ so that $\text{MSA}(Z)$ and Z have the same dimensionality. More details about transformers can be found in [66], [69].

3.3 Multiple MP3 Compression Localization

In this section we first introduce the temporal splicing detection and localization problem. We then present our proposed solution.

3.3.1 Problem Formulation

In MP3 compression, each audio signal is composed by a series of nonoverlapping fixed temporal length segments known as frames. Let x be the audio signal $x = \{x_1, x_2, \dots, x_L\}$, where x_l is the l -th frame of the signal, and L is the total number of frames, which depends on the signal length.

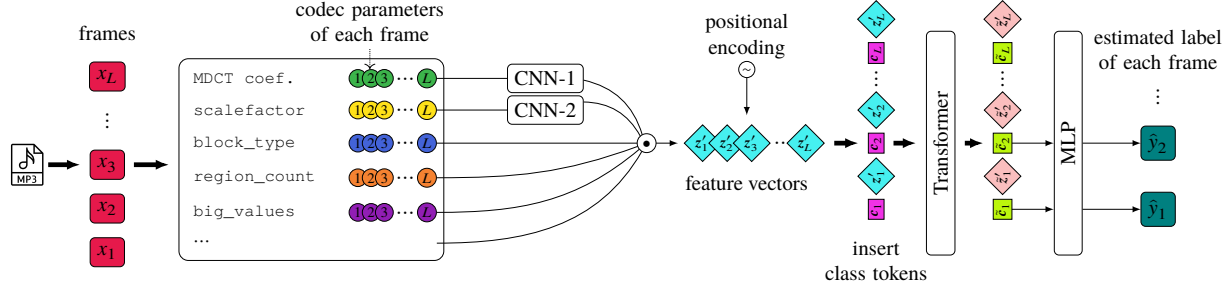


Figure 3.2. The block diagram of the proposed method, which analyzes L frames at a time. Each circular node (\odot) represents a MP3 codec parameter vector whose corresponding frame is shown by the number inside. Each diamond node (\diamond) represents a vector associated with the feature vector z'_l . Each rectangular node (\square) represents a vector associated with the class token c_l . Different groups of vectors are shown in different colors.

We can associate to the audio file a sequence of labels \mathbf{y} defined as $\mathbf{y} = \{y_1, y_2, \dots, y_L\}$, where y_l is the l -th binary valued label indicating whether the l -th frame (x_l) has been compressed once (i.e., $y_l = 0$) or more than one time (i.e., $y_l = 1$).

During the creation of a spliced MP3 audio file, it is likely that frames from different audio signals are concatenated in time and compressed using MP3. Some of the audio signals can be uncompressed (e.g., pristine or generated from a synthetic speech), while others may have been compressed and then decompressed. The final spliced signal will likely contain both single and multiple compressed frames.

Our goal is to find $\hat{\mathbf{y}}$, which is an estimate of the sequence of labels \mathbf{y} associated with the audio file \mathbf{x} by examining the MP3 compressed data for \mathbf{x} on a frame by frame basis. In doing so, we are able to detect if an audio file has undergone splicing, and localize which frames have been compressed more than one time.

3.3.2 Proposed Method

Our proposed method is shown in Figure 3.2. We examine the MP3 data corresponding to the individual frames of the input signal $\mathbf{x} = \{x_1, x_2, \dots, x_L\}$. We examine L frames at a time to decide if the audio signal has been multiply compressed and localize the splicing. A MP3 compressed frame consists of two groups of samples known as granules [56], [57]. Each granule contains 576

temporal samples from the respective two stereo channels. Our proposed method uses the data produced by the MP3 codec for a frame shown in Table 3.1 from the first channel of the first granule. This data consists of MDCT coefficients, quantization step sizes, scalefactors, Huffman table indices, and sub-band window selection information. The MP3 codec data we choose contain a complete set of parameters required to decode the channel. We will use this information to decide if an audio signal has been multiple compressed and localize the multiple compressed frames.

The `mdct_coef` and `scalefactor` fields are used as inputs to two separate Convolutional Neural Networks (CNNs) (i.e., CNN-1 and CNN-2) to find more features (see Figure 3.2). We use CNN architectures that are similar to the well known VGG CNN [72]. The depth and number of filters are changed based on the size of the `mdct_coef` and `scalefactor` fields. By denoting the convolutional layer as Conv<receptive field size>–<number of filters> and the fully connected layers as FC–<number of neurons>, the architecture and parameters of each CNN are as follows:

- CNN-1: Conv3-32, Conv3-32, Maxpool, Conv3-64, Conv3-64, Maxpool, Conv3-128, Conv3-128, Maxpool, FC-233, Dropout, FC-233, Dropout.
- CNN-2: Conv3-16, Conv3-16, Maxpool, Conv3-32, Conv3-32, Maxpool, Conv3-64, Conv3-64, Maxpool, FC-49, Dropout, FC-49, Dropout.

Table 3.1. MP3 codec information fields used in our method. Details about each field can be obtained from [73], [74].

Fields	Description
<code>part_23_length</code>	Size of coded binary data
<code>scalefactor, scalefac_compress, scalefac_scale, preflag</code>	Scalefactor value info
<code>global_gain, subblock_gain, big_values, region_count</code>	Quantization step sizes
<code>table_select, count1_table</code>	Huffman table selection info
<code>block_type, mixed_block_flag</code>	Sub-band window selection info
<code>mdct_coef</code>	Decoded MDCT coefficients

For the l -th frame, we concatenate the outputs of CNN-1 and CNN-2 as well as the values of the remaining fields into a feature vector $z_l \in \mathbb{R}^{d_{\text{model}}}$, which is used as the input to the transformers. The

sizes of the CNN outputs are selected so that the dimensionality of z_l 's satisfies $d_{\text{model}} = 300$. The self attention mechanism does not use the order information of the elements in the input sequence. To allow the transformer to make use of the temporal order of frames, the frames' corresponding feature vectors are added with positional encoding [66]. The positional encoding is a series of L distinct fixed-valued vectors $\{p_1, p_2, \dots, p_L\}$, where $p_l \in \mathbb{R}^{d_{\text{model}}}$. After adding positional encoding, the l -th feature vector is $z'_l = z_l + p_l$. The transformer will likely be able to find the position of z'_l being l based on the p_l component of z'_l .

We use a similar approach as described in [69] to binary classify the feature vectors corresponding for each MP3 frame as being “single compressed” or “multiple compressed”. The network has L special vectors, known as “class tokens”, which are denoted by $\{c_1, c_2, \dots, c_L\}$, where $c_l \in \mathbb{R}^{d_{\text{model}}}$. The input to the transformer Z is the feature vector z_l 's interleaved with the class tokens, which can be written as

$$Z = [c_1 \mid z'_1 \mid c_2 \mid z'_2 \mid \dots \mid c_L \mid z'_L] \in \mathbb{R}^{2L \times d_{\text{model}}}. \quad (3.5)$$

Let $\text{tf}(\cdot)$ be the function corresponding to the transformer network. After the transformer operations, the output can be written as

$$\text{tf}(Z) = [\tilde{c}_1 \mid \tilde{z}'_1 \mid \tilde{c}_2 \mid \tilde{z}'_2 \mid \dots \mid \tilde{c}_L \mid \tilde{z}'_L] \in \mathbb{R}^{2L \times d_{\text{model}}}. \quad (3.6)$$

To find the estimated labels for the l -th frame, we use a Multilayer Perceptron (MLP) network to classify \tilde{c}_l . During training, the gradients with respect to c_l 's are computed and used to update them using gradient descent [75]. That is, we train each class token to collect information necessary to determine the label for its corresponding time step. Finally, the label of each frame can be determined by only examining the class tokens.

After preliminary experiments, we used 8 transformer layers with the number of heads $h = 15$ in the Multihead Self Attention (see Section 3.2.2), which yielded the best performance.

The MLP network we use is made up of one layer of 800 neurons and a final output layer with 2 neurons using softmax activation. We use the GELU [76] nonlinear activation function in the CNNs and the transformer network.

After training, for a given MP3 audio frame sequence $\{x_1, x_2, \dots, x_L\}$, our method generates a binary decision sequence $\{\hat{y}_1, \hat{y}_2, \dots, \hat{y}_L\}$ describing whether each frame is multiply compressed or not. This enables one to localize the questionable frames in an MP3 file and determine which part of the MP3 frames may have been spliced.

3.4 Experiments and Results

In this section we describe our experiments and present results comparing our method to other methods for splicing localization using traces of multiple MP3 compression.

3.4.1 Dataset

The data for our experiments contain only uncompressed WAV audio files. We used three publicly available datasets consisting of speech and different music genres: LJSpeech [77], GTZAN [78], and MAESTRO [79]. We then compressed the WAV files using MP3. The MP3 sampling rate we selected is 44.1 kHz, and the length of each frame is $1152/44100 \approx 26.12\text{ms}$. In our experiments, our method examined $L = 20$ frames at a time.

The MP3 compression data rates and data rate types used in our experiments are shown in Table 3.2. We used Constant Bit Rate (CBR) and Variable Bit Rate (VBR) compression. The audio signals were compressed using FFmpeg¹v3.4.8 and LAME²v3.100. We excluded low-quality MP3 compression configurations that make multiple-compression detection unreliable [63]. We also excluded ultra-high-quality MP3 compression that is not recommended by FFmpeg [80].

Table 3.2. MP3 compression types used in our experiments. The data rate of VBR compression decreases as quality index increases. More details can be obtained from [80].

Compression type	Bit rate/Quality
Constant Bit Rate (CBR)	64kbps, 96kbps, 128kbps, 160kbps, 192kbps, 256kbps
Variable Bit Rate (VBR)	1, 2, 3, 4, 5, 6

¹<https://www.ffmpeg.org/>

²<https://lame.sourceforge.io/>

Our dataset generation scheme is described as follows. We split each uncompressed WAV audio file into smaller segments of 80–320 frames randomly. Each segment is further temporally subdivided into “slices” of $L/2 = 10$ frames where L is the number of frames our method examines at a time. Now, each segment will contain 8–32 slices. In each segment, the odd slices will be multiple compressed for 2–3 times. The even slices will be single compressed. The compression parameters are chosen from Table 3.2 at random. In Figure 3.3 we illustrate the dataset generation method on a segment containing 4 slices. If a slice needs to be compressed three times, then compression 1–3 will be used. If a slice needs to be compressed two times, then compression 2–3 will be used. Otherwise, only compression 3 will be used. For a given slice, after compression 1 the MP3 file is decompressed back to the time domain, which is used as the input to compression 2. At compression 3, we gather the decompressed or pristine time domain samples of all slices in a segment and then compress them as one MP3 file. Therefore, the parameters for compression 3 will be the same for all slices in a given segment. This completes the construction of our ground-truthed experimental data set.

	Slice 1 $L/2 = 10$ frames	Slice 2 $L/2 = 10$ frames	Slice 3 $L/2 = 10$ frames	Slice 4 $L/2 = 10$ frames
Compression 1	CBR, 256kbps			
Compression 2	VBR, Q=4		CBR, 160kbps	
Compression 3	CBR, 64kbps	CBR, 64kbps	CBR, 64kbps	CBR, 64kbps
$y = [1,1,1,1,1,1,1,1,1,1, 0,0,0,0,0,0,0,0,0,0, 1,1,1,1,1,1,1,1,1,1, 0,0,0,0,0,0,0,0,0,0]$				

Figure 3.3. An illustration of our dataset generation method on a segment of 40 frames (4 slices). The corresponding label for this segment y is shown at the bottom.

3.4.2 Preparing an MP3 file For Training

Recall our method operates in the “MP3 domain” by examining the MP3 codec fields shown in Table 3.1. To train our method, we used a sliding window of length L with offset step size 8 to extract the MP3 codec fields from the MP3 compressed frames as shown in Table 3.1.

We generated 486,743 MP3 frame sequences of length L from our experimental dataset. We used 54% of the frame sequences in the training set; 13% of the frame sequences in the validation

set, and 33% of the frame sequences in the testing set. The partition was done in a way that all frame sequences generated from one segment can only be used in one of the three sets.

Selecting the MP3 frame sequence length L for analysis is a trade-off between the accuracy of the estimation of \hat{y} and the training difficulty of the network. Using a larger L may improve the performance, but it may also significantly increase training time and the need of hyperparameter tuning. In our experiments, we used $L = 20$ corresponding to an audio signal length of approximately 522.4ms. This may be small compared to the typical length of most audio signals. In training we used a sliding window stride of 8, which does not align with the slice boundaries. In each sliding window, the index of the first multiple compressed frame and the number of multiple compressed frame are different. This forces the trained network to predict the labels correctly given an arbitrary portion taken from an audio signal. Therefore, our method is able to examine longer audio files even if L is relatively small.

3.4.3 Hyperparameters and Training

The entire network including the CNNs and the transformers are trained end-to-end. We trained the network using the Adam optimizer [81] with an initial learning rate of 10^{-4} and a dropout rate of 0.2 until the validation accuracy no longer increased for 20 epochs.

3.4.4 Results

We compared the performance of our method to [54], [59], [63], which are introduced in Section 3.1. Since they are all detection methods, we used the predictions from these methods on short frame sequences of length $L' = 4$ to approximate localization. Choosing the length L' for localization approximation is a trade-off between classification accuracy and granularity of localization. Larger L' improves the label estimation of each frame sequence, while smaller L' improves the localization accuracy. Selecting $L' = 4$ is a reasonable balance between these two factors [64].

In Table 3.3, we compare the performance of our method to that of the three previous methods. We used three different metrics: Jaccard score [82], F_1 -score [83] and balanced accuracy score [84]. Let $\mathcal{I}(\mathbf{y})$ be the function that returns the set of indices for the one-entries in \mathbf{y} . The Jaccard score can be computed by $|\mathcal{I}(\mathbf{y}) \cap \mathcal{I}(\hat{\mathbf{y}})| / |\mathcal{I}(\mathbf{y}) \cup \mathcal{I}(\hat{\mathbf{y}})|$, which compares the similarity between the predicted

multiple compressed region and the ground truth multiple compressed region [82]. The F_1 -score is the harmonic mean of the precision and recall, which considers both factors at the same time [83]. The balanced accuracy score is the traditional accuracy score with class-balanced sample weights [84]. Our approach achieved the highest score on all three metrics. We also tested our method on a separately generated dataset containing 53,541 MP3 frame sequences with variable slice length ranging from 10 to 80 frames. For each slice, the slice length, compression types and the number of compressions are chosen at random. Our method achieved 81.64 balanced accuracy on this dataset, which is close to the result in Table 3.3. This shows our method did not learn strong dataset bias.

In Table 3.4, we show the recall [83] of each method on multiply compressed MP3 frames against selected last MP3 compression types. It can be seen that the detection accuracy decreases as the MP3 compression quality declines. Different methods reacted to CBR and VBR compression in contrasting manners. For [54] and our method, the performance was similar. For [59], the score of VBR frames was significantly lower than those of CBR; the behavior of [63] was the opposite.

In table 3.5, we show the recall of each method compared to the number of MP3 compression applied to a frame. The recall of our method is more consistent across different repetition of MP3 compression. For all methods, the recall for double compression is close to that of triple compression.

Overall, our approach demonstrated high localization performance and robustness across many types of MP3 compression.

Table 3.3. Performance metrics comparison.

Method	Jaccard Score	F_1 -score	Balanced Accuracy
Yan, Wang, Zhou, Jin, and Wang [63]	13.53	18.71	53.35
Yang, Shi, and Huang [59]	30.73	40.95	55.28
Liu, Sung, and Qiao [54]	48.18	58.91	68.72
Our Approach	80.50	84.43	84.49

3.5 Conclusions

We proposed a multiple MP3 compression temporal localization method based on transformer neural networks that uses MP3 compressed data. We used two CNNs to extract features from

Table 3.4. The recall of multiple compression localization of each method against selected last MP3 compression types. CBR compression is denoted by C<bit rate>; VBR compression is denoted by V<quality index>.

Method	Last MP3 Compression Type							
	C64	C128	C160	C192	V1	V2	V4	V6
Yan, Wang, Zhou, Jin, and Wang [63]	17.30	6.86	6.80	4.87	22.80	23.23	22.59	17.55
Yang, Shi, and Huang [59]	19.27	70.75	71.71	66.21	45.58	39.05	27.12	22.06
Liu, Sung, and Qiao [54]	58.45	56.48	54.47	55.01	55.15	56.41	57.47	67.93
Our Approach	73.09	88.06	89.65	90.84	93.31	91.21	83.52	65.51

Table 3.5. The recall of each method against the number of MP3 compression.

Method	Num. MP3 Compression			
	Single	Double	Triple	Overall
Yang, Shi, and Huang [59]	67.28	43.51	43.02	43.27
Yan, Wang, Zhou, Jin, and Wang [63]	91.71	14.86	15.10	14.98
Liu, Sung, and Qiao [54]	79.36	57.27	58.85	58.06
Our Approach	84.61	83.76	84.92	84.34

the MP3 codec parameters `mdct_coef` and `scalefactor`. The transformer network generates temporal localization results by analyzing the relationship between the features corresponding to the MP3 compressed frames. Our proposed method localizes multiple compression at the frame level. We trained and tested our method on a dataset consisted of both CBR and VBR MP3 compression. The experiment results showed that our method had the best performance compared to other approaches and was robust against many MP3 compression settings. In the future, we will examine extending this approach to other compression methods such as AAC. We will also investigate the use of stereo channels as well as the second granule in MP3 compressed frames. We will generalize the concept of multiple compression detection to compression history detection. That is, to find the number of compressions and the types of compression used. Knowing the compression history can greatly enhance the interpretability for audio forensics.

4. H4VDM: H.264 VIDEO DEVICE MATCHING

4.1 Introduction

Video Device Identification (VDI) is one of the most important tasks in multimedia forensics [85], [86]. A VDI method can associate a video with a specific source device (e.g., a specific camera). VDI is valuable in forensics investigations and court defense.

A large amount of VDI techniques rely on the analysis of video camera sensor fingerprints. For example the Photo Response Non-uniformity (PRNU) pattern is commonly used in image/video device identification techniques [87]–[90]. Since PRNU patterns can capture the heterogeneity of the sensor response caused by imperfections in the sensor manufacturing process, such sensor fingerprints can attribute a video sequence to one source device uniquely. Despite being powerful in many forensics tasks, the PRNU patterns can be difficult to obtain. The estimation of PRNU patterns usually requires many samples taken by the device under analysis [86]. Obtaining the PRNU patterns from video sequences is more challenging due to the existence of video compression and video stabilization [91]. These challenges limit the application of PRNU patterns in VDI tasks.

H.264 is one of the most popular video compression techniques [92], [93]. It offers a wide variety of compression configurations to balance data rate, distortion, and computational complexity. Each configuration will induce a distinct encoding response to the input video sequence. Even for the same compression configuration, the behavior of H.264 encoder implementations used by different device manufacturers may not be identical. Note that the H.264 compression standard only standardizes the decoding but not the encoding [92], [93]. In this paper we shall refer to the “encoding pattern” of a H.264 compressed video sequence as the parameters inserted into the compressed bitstream by the encoder and used by the decoder to reconstruct the video sequence. We will show in this paper that these encoding patterns along with the video content can be used to determine the source device of an H.264 video sequence. Since the H.264 encoding pattern tends to be the same for a specific video camera model or firmware version, using it for VDI tasks may only result in a model-level or firmware-level match, which is coarser compared to techniques using video camera sensor fingerprints. However, these H.264 encoding patterns are closely related to the compressed digital video and are less affected by operations that alter the sensor fingerprints such as video stabilization. Compared to metadata forensic methods such as [94]–[96], video

forensics methods based on using the video content and encoding parameters can still work even if the metadata information is modified (e.g., when an MP4 video file is converted to an AVI video file without transcoding the video stream). Our proposed method only requires two H.264 Group of Pictures (GOPs) to make decisions, which allows our approach to work with corrupted or fragmented H.264 data.

In this paper, we focus on open-set Video Device Matching (VDM), which is related to VDI. Open-set classification is the problem of handling classes that are not contained in the training dataset. Traditional classification approaches assume that only known classes appear in the testing environment [97]. In VDI, the video forensics method is asked to identify which device was used to capture the video sequence. In VDM, the video forensics method is asked to determine if two video sequences are captured by the same camera model [98]–[100]. With VDM methods, VDI can be achieved by obtaining a video sequence from a known device and then determining if the video sequence under analysis is captured by the same device. Since VDI methods attribute a given video sequence to a specific source device, these methods require prior knowledge about the devices. As a result, VDI methods are often constrained to closed-set problems, where the video sequence under test comes from a device that is already known to the video forensics method [98]. This tends to have limitations in practice, as forensic investigators are more likely to deal with open-set problems where the video camera model has not been encountered by the method before. Because VDM methods only analyze if the two video sequences are from the same device, it is possible for these methods to work in open-set scenarios where the devices have never been seen by the VDM methods before [98]. We tested the open-set performance of our proposed VDM technique and verified that it had good performance evaluation metrics for unseen device models. The ability to attain open-set VDM allows our method to be used in a wider range of forensic investigations.

The rest of the paper is organized as follows. In Section 4.2, we show the related work and provide the background knowledge about the H.264 video compression and the machine learning model used in our approach. In Section 4.3, we describe the details of our proposed VDM method. In Section 4.4, we discuss the details of our experiments and present the results. Section 4.5 concludes the paper and gives insights on open problems and future challenges.

4.2 Background

In this section we show existing work related to Video Device Identification (VDI), Video Device Matching (VDM), and H.264-based video forensics. Then, we briefly introduce H.264 video compression and transformer neural networks, which are two important concepts for understanding the mechanism of our proposed open-set VDM method.

4.2.1 Related Work

Video Device Identification (VDI) is an important topic in video forensics. In [101], the authors used the statistics of motion vectors from video codecs for VDI. Yahaya *et al.* [102] used conditional probability features to achieve VDI. In [94]–[96], the authors used metadata information stored in video container formats for VDI. Most existing work on VDI are based on camera sensor noise fingerprints, which was first studied in [85]. In the seminal work proposed by Chen *et al.* [103], PRNU analysis was first used for VDI tasks. In [90], [99], [104]–[106], the authors devised various strategies to improve the performance of PRNU analysis of video, such as selecting key frames, counteracting the effects of video stabilization, using the characteristics of the video codecs, and weighting frames in terms of compression quality. Beyond PRNU patterns, [107]–[109] used deep neural networks to extract features from decoded video frames, which can be used in VDI tasks. Some VDI techniques use multimodal data to improve the performance. Iuliani *et al.* [89] combined image and video data from the same sensor for better VDI results. Dal Cortivo *et al.* [110] proposed a VDI approach based on video and audio information.

As described in Section 4.1, Video Device Matching (VDM) is a concept derived from VDI. It is also known as Video Device Verification. In [99], [100], the authors improved PRNU analysis for VDM. Mayer *et al.* [98] addressed the problem of open-set VDM using a deep neural network to extract features from decoded video frames.

There have been a number of video forensics approaches that use the characteristics of H.264 video compression. In [25], [111]–[117], the authors used information from the H.264 codec to determine if an H.264 video is double compressed. Verde *et al.* [118] used deep neural networks to extract H.264 codec information for video manipulation localization. We believe our proposed technique is the first to use H.264 codec information for open-set VDM problems.

4.2.2 H.264 Video Compression

The details of the H.264 video compression standard are extremely sophisticated and beyond the scope of this paper. Therefore, we provide a high-level overview of H.264 video compression that is sufficient for understanding our proposed VDM method. More details about H.264 compression can be obtained from [92], [93], [119].

One important concept that is a part of nearly all video compression standards is the use of both spatial and temporal redundancy in a video sequence to reduce its data rate, particularly the fact that consecutive frames in a small temporal interval can be greatly correlated. The H.264 encoder examines the video frames in a structure known as Group of Picture (GOP), which is a sequence of consecutive frames. Due to the high temporal correlation, the frames in a GOP can be compressed using motion compensation [93]. The frames in a GOP are divided into I-, P-, and B-frames that are used for the motion compensation. Typically the first frame in a GOP is an I-frame that acts as the reference frame for other frames in the GOP. The I-frame is compressed using intra-frame compression similar to JPEG. The rest of the frames in an H.264 GOP can be P-frames or B-frames, and are compressed with inter-frame compression and motion compensation using the I-frame or a P-frame as a reference frame [93].

In I-, P-, and B-frames, H.264 uses 16×16 frame patches known as macroblocks in each video frame. Each macroblock is associated with a macroblock type, which specifies how the information in the macroblock is compressed. Macroblock-level compression can be done by predicting patterns within the macroblock (i.e., intra-coded macroblocks) or using difference information from a similar macroblock in a motion compensated reference frame (i.e., inter-coded macroblocks) [93]. This difference is known as the prediction residual, which is more efficient to compress compared to the original video frame. The pixels in the macroblocks are then transformed and quantized before being entropy coded [93]. The quantization process is controlled by a Quantization Parameter (QP). As the QP increases, the data rate decreases and the quantization distortion increases; as QP decreases, the rate-distortion trade-off goes towards the opposite direction [93]. For inter-coded macroblocks the encoded prediction residuals and the motion vectors from the motion compensation are placed in the compressed bitstream.

H.264 uses the YUV color space, which has one luma (luminance) channel and two chroma (chrominance) channels [93]. Since the human visual system is more sensitive to luminance than chrominance, H.264 prioritizes the compression quality of the luma channel over the chroma channels to reduce data rate. Each color channel has its own QP for rate-distortion control [93].

4.2.3 Vision Transformers

Our proposed VDM method is based on transformer neural networks [66]. These networks demonstrated outstanding performance in a wide variety of tasks such as language modeling [68], image classification [69], image segmentation [120], video classification [121], audio signal processing [122], and protein structure prediction [71].

Transformers can be used to process sequence data. They possess faster computational speed, higher scalability, and better stability compared to Recurrent Neural Networks such as LSTM [123] and GRU [124], [125]. Transformer networks are made up of transformer layers. The output of a transformer layer is used as the input to the next transformer layer. Denote the input to a transformer layer by $\mathbf{Z} \in \mathbb{R}^{N \times D}$, where N is the sequence length and D is the dimensionality of the vector at each time step. Transformer layers estimate the relationship between the vectors at each time step in \mathbf{Z} based on the Self Attention (SA) mechanism. In SA, the input \mathbf{Z} is first linearly projected into three matrices $\mathbf{q}, \mathbf{k}, \mathbf{v} \in \mathbb{R}^{N \times D_h}$ using a matrix $\mathbf{U} \in \mathbb{R}^{D \times 3D_h}$ such that $[\mathbf{q}|\mathbf{k}|\mathbf{v}] = \mathbf{Z}\mathbf{U}$. Then, the SA of \mathbf{Z} is computed by $\text{SA}(\mathbf{Z}) = \text{softmax}(\mathbf{q}\mathbf{k}^T/\sqrt{D_h})\mathbf{v}$. An extended version of SA known as Multihead Self Attention (MSA) is used in [66], where h different SA values (known as “MSA heads”) are computed at the same time. The h different SA values are combined to form the result of MSA using a matrix $\mathbf{V} \in \mathbb{R}^{D \times D}$ such that $\text{MSA}(\mathbf{Z}) = [\text{SA}_1(\mathbf{Z}) | \cdots | \text{SA}_h(\mathbf{Z})]\mathbf{V}$. When choosing the number of MSA heads h , it must hold that h divides D and $D_h = D/h$, which ensures the output of the transformer layer $\text{MSA}(\mathbf{Z})$ has the same dimensionality as the input \mathbf{Z} . Since a transformer network consists of a series of such layers, its input and output share the same dimensionality.

Vision Transformer (ViT) [69] uses transformer networks for computer vision tasks. In order to convert image data into the format accepted by transformers, the ViT splits the image into K non-overlapping patches of size $P \times P \times C$, where P is the patch size and C is the number of channels. The patches are flattened to form the matrix $\mathbf{Z}' \in \mathbb{R}^{K \times P^2 C}$. Since $P^2 C$ is usually large, the

dimensionality of the flattened vectors from each patch is reduced to D_{ViT} using a linear projection $\mathbf{W} \in \mathbb{R}^{P^2 C \times D_{\text{ViT}}}$ such that $\mathbf{Z} = \mathbf{Z}'\mathbf{W}$. The original input image can be represented by \mathbf{Z} , which is used as the input to the transformer network.

4.3 Proposed Method

The H.264 Video Device Matching (VDM) problem can be formally defined as follows: given two Group of Pictures (GOPs) \mathcal{G}_1 and \mathcal{G}_2 from two video sequences, determine if the two GOPs are from the same device. Note the GOP information includes the decoded frame and the coding parameters. We propose an H.264 Video Device Matching (H4VDM) method for open-set VDM. The block diagram of H4VDM is shown in Figure 4.1. This network architecture is commonly used in many open-set digital forensics techniques [88], [98], [126]. The input to H4VDM is two H.264 GOPs \mathcal{G}_1 and \mathcal{G}_2 . They are then passed to the H4VDM feature extractor separately.

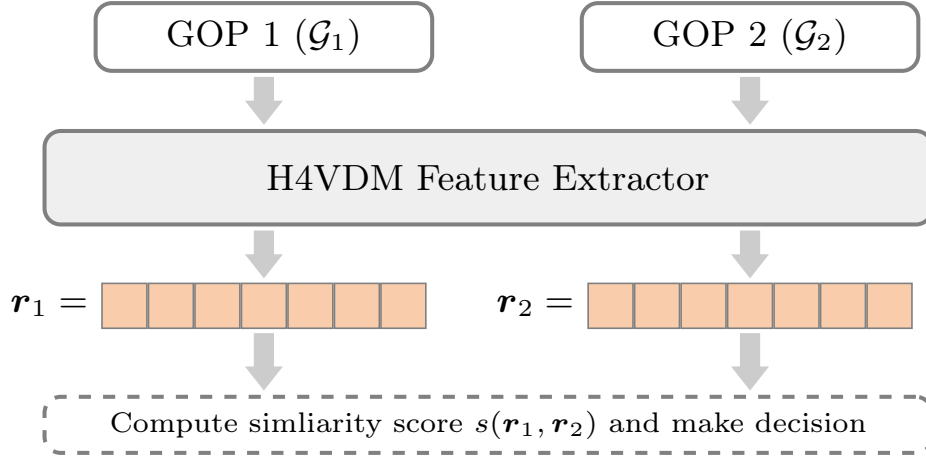


Figure 4.1. The block diagram of our proposed H.264 Video Device Matching (H4VDM) method. The details of the H4VDM feature extractor are described in Section 4.3.1. Note the input GOP information includes the decoded frame and the coding parameters.

The H4VDM feature extractor is the key component of our proposed method, which is described extensively in Section 4.3.1. It extracts important information from an H.264 GOP \mathcal{G} and expresses this information in a D_r -dimensional vector representation \mathbf{r} known as the GOP feature vector. Let \mathbf{r}_1 and \mathbf{r}_2 be the corresponding GOP feature vectors of \mathcal{G}_1 and \mathcal{G}_2 , respectively. We compute a similarity score $s(\mathbf{r}_1, \mathbf{r}_2)$ between the two GOP feature vectors. The similarity score can be used for

classification: a higher similarity score indicates that \mathcal{G}_2 is more likely to be captured by the same device as \mathcal{G}_1 . The similarity score used in our proposed method is described in Section 4.3.2.

4.3.1 H4VDM Feature Extractor

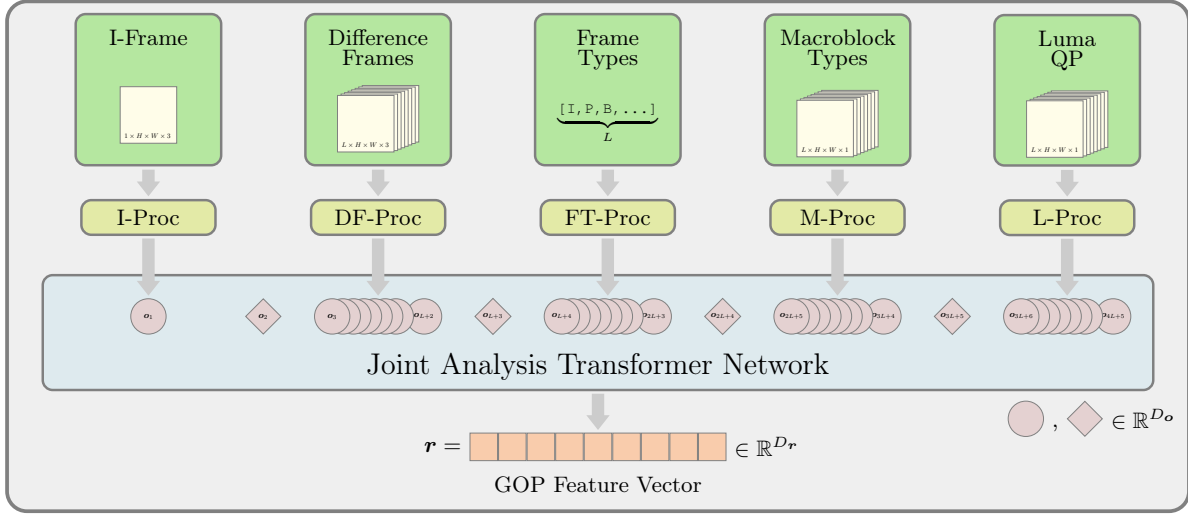


Figure 4.2. The block diagram of the H4VDM feature extractor.

The block diagram of the H4VDM feature extractor is shown in Figure 4.2. The feature extractor computes GOP feature vectors using H.264 GOPs containing L frames, where the size of each frame is $H \times W$. If a GOP is longer than L frames, we extract features for the first L frames only without loss of generality. If the frame size of a GOP is larger than $H \times W$, we extract features from an arbitrary $H \times W$ region in the GOP for our analysis. Feature extraction is not possible when the length of GOP is less than L or when the GOP frame size is smaller than $H \times W$. Therefore, it is important to set L , H , and W appropriately so that the H4VDM feature extractor can process GOPs from a wide range of H.264 video sequences.

The H4VDM feature extractor uses five types of data from an H.264 GOP to generate GOP feature vectors, i.e., the I-frame, the frame differences, the frame types, the macroblock types and the luma QPs. Each type of data is first processed by a specific approach to generate an intermediate output denoted by $o \in \mathbb{R}^{D_o}$. The five processing methods for each data type are known as the I-, DF-, FT-, M-, and L-Proc. Many of these processing methods are using Vision Transformers (ViTs) to process image-like information. In total, we use two ViT architectures in our method, i.e., ViT-1

and ViT-2 (as shown in Table 4.1). ViT-1 is a larger network that processes more complicated data such as I-frames and residual frames. ViT-2 is a smaller network that processes simpler data such as macroblock types and luma Quantization Parameters (QPs). The details of each processing method are described as below.

4.3.1.1 I-Proc.

This processing step extracts feature from the decoded I-frame in the GOP. The I-Proc uses the ViT-1 architecture in Table 4.1. For each GOP, the input to the I-Proc is an $H \times W \times 3$ vector consisting of the RGB pixel data from the I-frame. The output of the I-Proc is \mathbf{o}_1 .

4.3.1.2 DF-Proc.

This processing step extracts features from differences of the decoded frames in sequence. The differences we compute are between the decoded frames and the decoded I-frame in the GOP (including the difference between the I-frame and itself, which is all zeros). Using difference frames can better enable the DF-Proc to learn about the characteristics of H.264 compression. The DF-Proc also uses the ViT-1 architecture in Table 4.1. For each GOP, the input to the DF-Proc is L vectors of dimension $H \times W \times 3$, where each vector is the RGB pixel difference. The L frame difference vectors are processed one by one to generate L outputs $\mathbf{o}_3, \dots, \mathbf{o}_{L+2}$.

4.3.1.3 FT-Proc.

This step converts frames types into D_o dimensional vectors. For each input GOP, the frame types are a sequence of L positive integers, each of which represents a valid frame type in H.264 (i.e., I, P, B). The FT-Proc projects each integer to a D_o -dimensional real-valued vector with the widely used embedding technique [127]. In the original integer representation, each integer is an index for a concept (e.g., frame types). The distance between two integer indices is not meaningful. By converting the integer indices into real-valued vectors using a projection learned in the training phase, the vector representation of similar concepts can have smaller distances, which makes learning easier for the machine learning method. The final outputs are L vectors $\mathbf{o}_{L+4}, \dots, \mathbf{o}_{2L+3}$.

Table 4.1. The hyperparameters of the vision transformers (ViT) [69] used in H4VDM, as discussed in Section 4.4.3.

Hyperparameters	ViT-1	ViT-2
depth	8	4
projection dimension	D_{ViT1}	D_{ViT2}
number of MSA heads	8	4
output dimension	D_o	D_o
patch size	16	16

4.3.1.4 M-Proc.

Here we extract features from the macroblock types. In the original H.264 data stream, a frame is subdivided into macroblocks of size 16×16 . Each macroblock in the frame can be compressed with different methods. The compression method used in a macroblock is stored as an integer known as the macroblock type [116]. This macroblock type information is converted into a $H \times W \times 1$ vector by “unpacking” the macroblocks. That is, every pixel in the frame is associated with a macroblock type integer inherited from the macroblock it belongs to. We use the embedding technique to project each macrotype integer into a three dimensional real-valued vector, which is then processed by a ViT-2 network described in Table 4.1. For the L frames in the GOP, the output is L vectors $\mathbf{o}_{2L+5}, \dots, \mathbf{o}_{3L+4}$.

4.3.1.5 L-Proc.

This step extracts features from the luma QPs. The input is an $H \times W \times 1$ vector, where each element is an integer ranging from 0 to 51, i.e., the luma QPs. As the luma QP increases, the H.264 quantization procedure discards more details in the luma channel in exchange for lower data rate [128]. Due to the ordered nature of luma QPs, we process them directly using a ViT-2 network in Table 4.1. For the L frames in the GOP, the output is L vectors $\mathbf{o}_{3L+6}, \dots, \mathbf{o}_{4L+5}$.

The intermediate outputs from the five processing networks contain important information about different data types. Similar to [129], we insert special vectors (i.e., $\mathbf{o}_2, \mathbf{o}_{L+3}, \mathbf{o}_{2L+4}, \mathbf{o}_{3L+5}$) to combine the information acquired from various data types. These special vectors can be updated during training. In total, there are $4L + 5$ intermediate output vectors, which are used as the input to

the joint analysis network. This network is an 8-layer transformer network [66]. The output of the joint analysis network is linearly projected to a vector $\mathbf{r} \in \mathbb{R}^{D_r}$. The vector \mathbf{r} is the output of the H4VDM feature extractor (i.e., the GOP feature vector).

Based on the five data types in the input GOP, the H4VDM feature extractor characterizes macroblock type selection, luma QP selection, and other patterns that are specific to the video capturing device. This information is contained in a D_r -dimensional GOP feature vector. By comparing the similarity score between the corresponding GOP feature vectors from two video sequences we can determine if the two video sequences were captured by the same device. Since H4VDM only requires two H.264 GOPs to make decisions, it is able to work in scenarios where data from a test device is scarce, e.g., when the H.264 video sequence is corrupted and only a few GOPs can be recovered.

4.3.2 Similarity Score and Loss Function

The similarity score between two GOP feature vectors \mathbf{r}_1 and \mathbf{r}_2 is a real number in the range $[0, 1]$, where 1 indicates the two vectors are the most similar and 0 indicates the two vectors are the most dissimilar. We compute the similarity score using the following function

$$s(\mathbf{r}_1, \mathbf{r}_2) = 1 - \tanh(\|\mathbf{r}_1 - \mathbf{r}_2\|_2), \quad (4.1)$$

where $\|\cdot\|_2$ denotes L_2 -norm. The function $f(x) = 1 - \tanh(x)$ for $x \geq 0$ is shown in Figure 4.3. When \mathbf{r}_1 and \mathbf{r}_2 are more similar, $\|\mathbf{r}_1 - \mathbf{r}_2\|_2 \rightarrow 0$, which indicates $s(\mathbf{r}_1, \mathbf{r}_2) \rightarrow 1$. Conversely, $s(\mathbf{r}_1, \mathbf{r}_2) \rightarrow 0$ when \mathbf{r}_1 and \mathbf{r}_2 are more dissimilar.

We use binary cross-entropy loss and the similarity score (Equation (4.1)) to compute the loss of the H4VDM method during training. Suppose the ground truth label of the GOP feature vector pair $(\mathbf{r}_1, \mathbf{r}_2)$ is given by y , where $y = 1$ indicates the two GOP features are from the same video capturing device and $y = 0$ indicates the opposite. The loss of the pair is

$$\ell(\mathbf{r}_1, \mathbf{r}_2, y) = y \log [s(\mathbf{r}_1, \mathbf{r}_2)] + (1 - y) \log [1 - s(\mathbf{r}_1, \mathbf{r}_2)]. \quad (4.2)$$

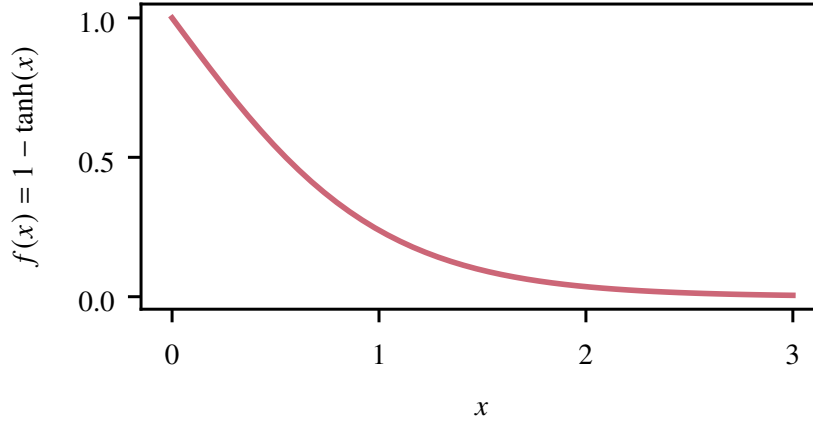


Figure 4.3. The similarity score function.

4.4 Experiments and Results

We describe our experiments in this section including the datasets used for training and testing. More details about H4VDM such as hyperparameter selection and training strategy are also discussed. Finally, we present the results from the experiments. We used Area Under the Receiver Operating Characteristic (AUC) score [130], F_1 -score [131], and accuracy score to evaluate the performance of H4VDM. Throughout the experiments, we selected the dimensionality of the GOP feature vectors to be 1024, i.e., $D_r = 1024$.

4.4.1 Dataset Generation

The datasets used in our experiments are generated from the VISION dataset [132]. The VISION dataset contains 648 H.264 video sequences from 35 different video capturing devices. The list of devices in VISION is shown in Table 4.2.

We decoded the video frames and extracted the GOPs from video sequences in VISION using a customized version of the `openh264`³ H.264 decoder, which allows us access to the GOP information. In our analysis, we selected the length of GOP $L = 8$. That is, our method can analyze H.264 GOP with length greater than or equal to 8. The height (H) and width (W) of the frames

³<https://github.com/cisco/openh264>

Table 4.2. The list of devices contained in the VISION dataset [132].

Device ID	Device Name	Device ID	Device Name
1	Samsung_GalaxyS3Mini	19	Apple_iPhone6Plus (iOS 10.2.1)
2	Apple_iPhone4s (iOS 7.1.2)	20	Apple_iPadMini (iOS 8.4)
3	Huawei_P9	21	Wiko_Ridge4G
4	LG_D290	22	Samsung_GalaxyTrendPlus
5	Apple_iPhone5c (iOS 10.2.1)	23	Asus_Zenfone2Laser
6	Apple_iPhone6 (iOS 8.4)	24	Xiaomi_RedmiNote3
7	Lenovo_P70A	25	OnePlus_A3000
8	Samsung_GalaxyTab3	26	Samsung_GalaxyS3Mini
9	Apple_iPhone4 (iOS 7.1.2)	27	Samsung_GalaxyS5
10	Apple_iPhone4s (iOS 8.4.1)	28	Huawei_P8
11	Samsung_GalaxyS3	29	Apple_iPhone5 (iOS 9.3.3)
12	Sony_XperiaZ1Compact	30	Huawei_Honor5c
13	Apple_iPad2 (iOS 7.1.1)	31	Samsung_GalaxyS4Mini
14	Apple_iPhone5c (iOS 7.0.3)	32	OnePlus_A3003
15	Apple_iPhone6 (iOS 10.1.1)	33	Huawei_Ascend
16	Huawei_P9Lite	34	Apple_iPhone5 (iOS 8.3)
17	Microsoft_Lumia640LTE	35	Samsung_GalaxyTabA
18	Apple_iPhone5c (iOS 8.4.1)		

were set to $H = W = 224$, which is a common size used by popular image or video processing techniques such as [69], [120], [121], [133]. With this choice of H and W , the frame sizes of all video sequences in the VISION dataset are larger than (H, W) . Therefore, we cropped a (H, W) region from the center of the frames for analysis. We chose to crop the region at the center because the video content at the center is more likely to change compared to those from the edges or corners. From each video sequence in the VISION dataset, we randomly sampled 15 GOPs whose length is greater than or equal to L . When the length of a sampled GOP is greater than L , only the first L frames in the GOP were used.

We constructed data from the VISION dataset to train and test our method as follows:

1. Provide a set of device indices \mathcal{S} , selected from Table 4.2.
2. Select all pairs of device indices $(i, j) \in \mathcal{S} \otimes \mathcal{S}$, where \otimes denotes Cartesian product. Denote the set of GOPs from device i and device j by \mathcal{A}_i and \mathcal{A}_j , respectively. For each (i, j) do the following:

- (a) If $i \neq j$, randomly sample n_0 unique GOP pairs from the set $\mathcal{A}_i \otimes \mathcal{A}_j$ that are not in the dataset. When determining if a GOP pair is in the dataset, the GOP pair is considered to be unordered. That is, if one swaps the first and the second element, the pair is considered to be the same. These n_0 pairs are assigned label 0 and added to the dataset.
- (b) If $i = j$, randomly sample n_1 unique GOP pairs from the set $\mathcal{A}_i \otimes \mathcal{A}_j$ that are not in the dataset. When determining if a GOP pair is in the dataset, the GOP pair is considered to be unordered. These n_1 pairs are assigned label 1 and added to the dataset.

In our experiments, we chose $n_0 = 15$ and $n_1 = 120$. To better evaluate the performance of H4VDM, we constructed 7 datasets, where each dataset contained a training set and a testing set. For brevity, we refer to these datasets as D1, D2, \dots , D7. In each dataset, the set of all device IDs in VISION are split into two disjoint sets \mathcal{S}_1 and \mathcal{S}_2 . They are passed to the dataset construction step to generate the training set and the testing set, respectively. For dataset D1–D4, \mathcal{S}_2 contained half of the device IDs that were randomly selected. For dataset D5–D7, the device IDs are split into three disjoint sets and used as \mathcal{S}_2 of each dataset. The details of each dataset are shown in Table 4.3. For the testing set of each dataset, we uniformly sampled 40 percent of the GOP pairs after dataset generation to reduce its size. In each dataset, we uniformly removed $1/8$ of the GOP pairs from the testing set for validation during training. The training process is stopped when the open-set validation performance no longer increases.

Table 4.3. The details of each dataset. #0 and #1 indicates the number of GOP pairs with class 0 (different device) and class 1 (same device), respectively.

Dataset	Test Device IDs (\mathcal{S}_2)	Training		Testing	
		#0	#1	#0	#1
D1	{1, 2, 4, 5, 6, 14, 17, 18, 19, 21, 22, 23, 27, 28, 30, 32, 35}	4590	2160	816	1632
D2	{1, 2, 4, 11, 14, 15, 17, 18, 19, 20, 21, 23, 26, 30, 32, 33, 35}	4590	2160	816	1632
D3	{4, 5, 6, 10, 11, 13, 14, 16, 17, 19, 21, 22, 23, 30, 31, 32, 35}	4590	2160	816	1632
D4	{3, 4, 6, 8, 13, 17, 19, 20, 21, 22, 23, 26, 29, 30, 31, 32, 34}	4590	2160	816	1632
D5	{1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34}	7590	2760	792	576
D6	{2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35}	7590	2760	792	576
D7	{3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33}	8280	2880	660	528

We selected $L = 8$ in our experiments based on several factors. Increasing L is likely to improve the quality of GOP feature extraction, because the H4VDM feature extractor is exposed to more

information. However, it will increase the complexity of the model and make the training process more computationally intense. Since our method requires the length of the GOP under analysis to be at least L , a larger L value can also reduce the number of valid GOP candidates in a video sequence. Note that it is computationally expensive to acquire the decoded frames and GOP information from the H.264 bitstream. In training, since the training data will be used repeatedly in each training epoch, it is more efficient to store the decoded frames and GOP information on disk. Therefore, another factor that affects the choice of L is the storage size of the uncompressed GOPs and decoded frames. With the current dataset configurations, each dataset (training set and testing set) requires approximately 176GB of disk space. Given such an enormous dataset size, the training process is heavily bounded by the I/O performance of the storage devices, which is much slower compared to other hardware components (e.g., GPUs). In the worst case, the training time can increase linearly with respect to L , which is costly when L is large. We used $L = 8$ as a balance among model performance, model complexity, the usability of our method, training complexity, and training speed. Typical training times were 3-6 hours using six 48GB GPUs.

4.4.2 Parameter Initialization and Training

From Figure 4.3 it can be seen that the gradients of the similarity score function $f(x)$ will saturate when x is large, which can significantly reduce the efficiency of gradient-based learning. Therefore, when initializing the model parameters, we limited the scale of the weights in the output layer to be within $[-0.002, 0.002]$. This reduced the initial scale of $\|r_1 - r_2\|_2$, which can facilitate training.

We trained the H4VDM method using the Adam optimizer [81] with a minibatch size of 72. We used 5 warm-up epochs, where the learning rate linearly increased from 0 to 8×10^{-6} . After warm-up epochs, we trained the method using an initial learning rate of 8×10^{-6} . An exponential learning rate decay was used with a decay factor of 0.97. During training, we monitored the AUC score on the validation set. The training was stopped when the validation AUC score no longer increased.

4.4.3 Model Size Selection

We tuned the hyperparameters D_{ViT1} and D_o to control the size of the model. We constructed 3 models of various sizes (i.e., S (small), B (baseline), and L (large)). For each model, we computed the best AUC score on the testing set of Dataset D1. The results are shown in Table 4.4. Since the H4VDM-B model achieved the best AUC score, further experiments were conducted with this model.

Table 4.4. The best AUC scores of various models on the testing set of Dataset D1.

Model	D_{ViT1}	D_{ViT2}	D_o	Parameters	AUC
H4VDM-S	192	64	192	48.93M	79.8
H4VDM-B	256	64	256	80.10M	80.2
H4VDM-L	320	64	320	118.95M	69.4

4.4.4 Results

4.4.4.1 Results on Datasets D1–D4.

In Table 4.5, we show the performance of the H4VDM-B model on datasets D1–D4. When computing the F_1 -score, we selected the threshold such that the sum of True Positive Rate (TPR) and True Negative Rate (TNR) is maximized. Our method achieved an overall F_1 -score of 67.2 and an average AUC score of 77.4 on datasets D1–D4. Overall, for class 0 (GOP pairs from different devices) the precision is high, which means most retrieved GOP pairs are relevant. For class 1 (GOP pairs from the same device) the recall is high, which means most relevant GOP pairs are retrieved. In Figure 4.4, we show the accuracy score matrix of device index pairs of each dataset in testing. From this figure, it can be seen that H4VDM can match devices at a firmware level. For example, device 29 and 34 in dataset D4 are the same device (Apple iPhone5) with different operating system versions. H4VDM was able to distinguish them with high accuracy.

Table 4.5. The testing performance of the H4VDM-B model on datasets D1–D4.

Dataset	Class 0			Class 1			All Classes			
	Pre.	Rec.	F_1	Pre.	Rec.	F_1	Pre.	Rec.	F_1	AUC
D1	95.9	53.6	68.8	50.6	95.4	66.1	80.8	67.5	67.9	80.2
D2	91.7	49.7	64.5	47.6	91.1	62.5	77.0	63.5	63.8	74.1
D3	89.9	55.4	68.5	49.4	87.5	63.2	76.4	66.1	66.8	78.0
D4	86.0	64.0	73.4	52.3	79.1	63.0	74.8	69.0	69.9	77.3
Overall	90.5	55.7	68.9	49.9	88.3	63.7	76.9	66.5	67.2	77.4

Pre.=Precision

Rec.=Recall

4.4.4.2 Results on Datasets D5–D7.

From the results of H4VDM on datasets D1–D4 (Figure 4.4), it can be seen that the performance of H4VDM was low for specific device pairs. This may be caused by the fact that datasets D1–D4 contain only a small number of devices in the training set, which makes it difficult for the method to generalize to a broader scope of devices. To test the performance of H4VDM on datasets with more training devices, we trained and tested H4VDM on datasets D5–D7, whose training sets contain more devices in the VISION dataset. The testing performance of the H4VDM-B model on datasets D5–D7 is shown in Table 4.6. The accuracy score matrix of device index pairs is shown in Figure 4.5. On datasets D5–D7, our method achieved an overall F_1 -score of 78.6 and an average AUC score of 85.2. It can be seen that as the number of devices in the training set increases, the performance of H4VDM becomes better. However, since the number of devices in the testing set decreases, the testing performance can vary significantly depending on the choice of the testing devices. Overall, the results on datasets D1–D4 show that H4VDM can learn to achieve open-set VDM given a small number of training devices. The results on datasets D5–D7 show that the performance of H4VDM can improve quickly as more devices are available in training.

4.5 Conclusion

In this paper we proposed an H.264-based open-set VDM method known as H4VDM. H4VDM uses transformer neural networks to process five types of data from the H.264 decoded frames and the GOPs. We trained and tested the H4VDM-B model on datasets generated from the VISION

Table 4.6. The testing performance of the H4VDM-B model on datasets D5–D7.

Dataset	Class 0			Class 1			All Classes			
	Pre.	Rec.	F_1	Pre.	Rec.	F_1	Pre.	Rec.	F_1	AUC
D5	85.9	74.5	79.8	75.5	86.5	80.6	80.9	80.2	80.2	87.0
D6	96.1	73.4	83.2	76.9	96.7	85.7	86.9	84.5	84.4	90.0
D7	94.8	48.8	64.4	65.5	97.3	78.3	80.1	73.0	71.3	78.5
Overall	92.3	65.6	75.8	72.6	93.5	81.5	82.6	79.2	78.6	85.2

Pre.=Precision Rec.=Recall

dataset [132]. The experimental results showed that H4VDM demonstrated good VDM performance on unseen devices.

Despite the good performance, H4VDM has still room for improvement. When selecting hyperparameters, we greedily used the model that had the best performance on Dataset D1. The selected model may not have the best overall performance across all datasets. Some important H.264 codec information such as motion vectors and true prediction residuals were not used in H4VDM. The small number of devices in the VISION dataset also limited the performance evaluation of H4VDM. On datasets D1–D4, less dataset bias is introduced in training/testing split, but the performance of H4VDM is relatively low due to small number of devices in the training set. On datasets D5–D7, the performance of H4VDM is higher, but the influence of dataset bias is stronger due to the small number of devices in the testing set, which resulted in fluctuating testing performance. A dataset with more devices is required to evaluate the performance of H4VDM more comprehensively.

In future work, we will examine the use of motion vectors and prediction residuals. We will develop methods that use popular deep learning frameworks efficiently so that the training speed is less constrained by hardware I/O speed. We will collect video data from more video capturing devices for future video forensics research. We are investigating other video compression techniques including H.265, H.266, VP9, and AV1.

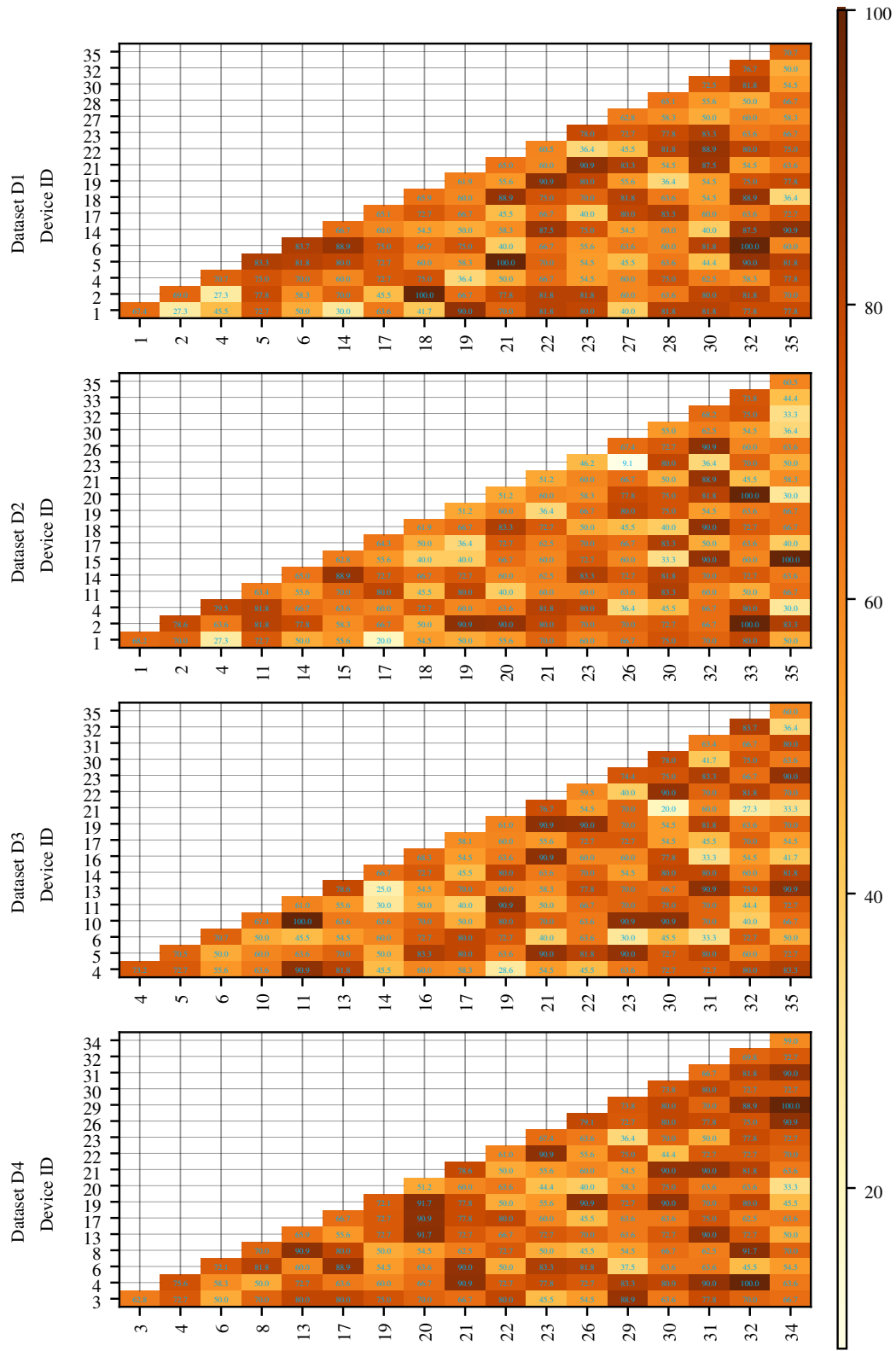


Figure 4.4. The testing accuracy score matrix of device index pairs on datasets D1–D4.

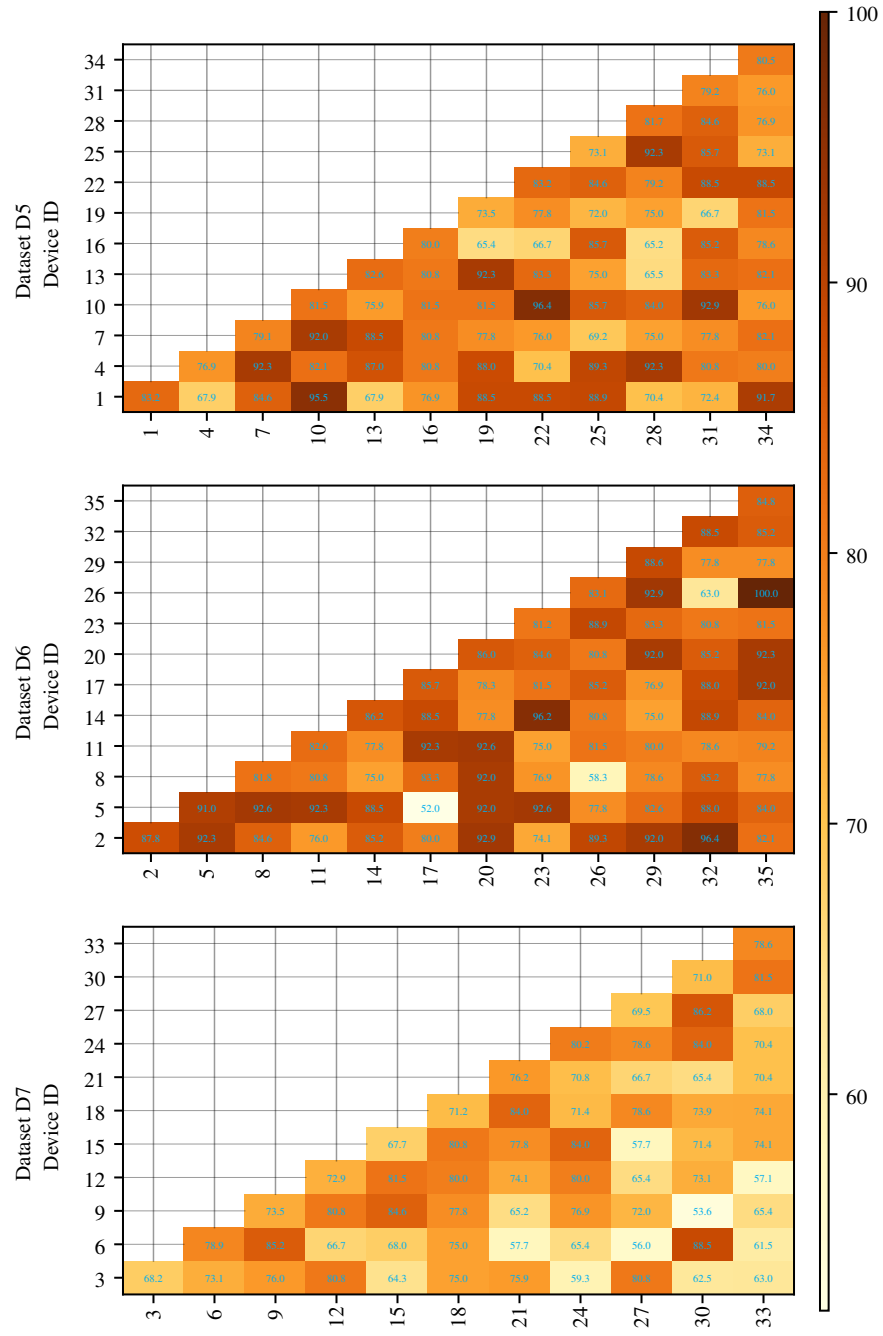


Figure 4.5. The testing accuracy score matrix of device index pairs on datasets D5–D7.

5. MTN: FORENSIC ANALYSIS OF MP4 VIDEO FILES USING GRAPH NEURAL NETWORKS

5.1 Introduction

The MP4 video container [134] is one of the most popular video container standards. MP4 files use a tree data structure to store information internally [134], [135]. In addition to video/audio bit streams which occupy most of the space in an MP4 file, other metadata information such as subtitles, codec type, video-audio synchronization information, timestamps, and geographic locations can also be stored. An MP4 file can be logically split into two components: 1) the “MP4 tree” that contains information related to both the tree topology and the metadata stored in the container; 2) the “bit streams” that contain information related to the encoded video pixel values and audio tracks. Existing work [94], [136]–[138] have shown that the MP4 tree (*i.e.* metadata information as well as the topology of the MP4 tree) can be used for video forensic analysis. This is a relatively new and developing concept since most existing Video Forensics Methods (VFMs) use the bit stream (*i.e.* pixel data) to make decisions in tasks such as camera model attribution, deepfake detection, and manipulation detection [139]–[141].

It has been shown in recent work that VFMs using the MP4 trees can provide an independent and alternative perspective compared to those that rely on pixel data (*i.e.* the bit streams). For example, Güera *et al.* [136] used metadata information extracted by the *ffprobe*⁴ tool with Support Vector Machines (SVMs) and decision trees for video manipulation detection. Iuliani *et al.* [135] used hand-crafted features from the MP4 tree with a statistical classifier for video forensics analysis tasks. Yang *et al.* [137] converted the MP4 tree to an integer vector representation, which can be processed by decision tree classifiers for video forensics analysis tasks. Xiang *et al.* [94] extended the approach in [137] by improving MP4 tree parsing and introducing feature selection and dimensionality reduction to the vector representation of MP4 trees. Altinisik *et al.* [138] fused MP4 tree and video bit stream analysis for video source attribution.

In this chapter, we propose MP4 Tree Network (MTN), which is a VFM based on end-to-end Graph Neural Networks (GNNs). MTN analyzes only the information in the MP4 tree (*i.e.* no bit stream data is used) and can be used to perform different forensic tasks.

⁴<https://ffmpeg.org/ffprobe.html>

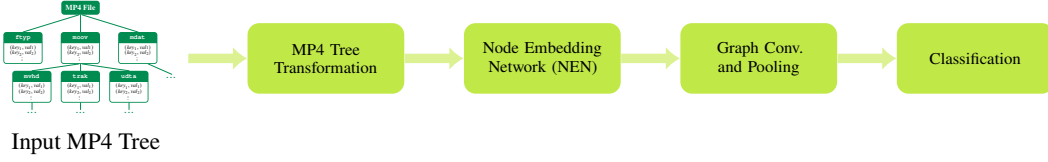


Figure 5.1. The block diagram of MP4 Tree Network (MTN).

In existing MP4 tree analysis [94], [136], [137], hand-crafted features with light-weight classifiers such as SVM [142] or decision trees [143] are used to predict the labels for the MP4 trees. Due to the limitations of hand-crafted features and light-weight classifiers, these methods are unable to process unseen data in MP4 trees and often fail to gain comprehensive understanding about the MP4 tree. MTN does not have these shortcomings as it is based on end-to-end deep neural networks. The scalability of deep neural networks also allows MTN to process very large datasets. We propose a Self-Supervised Learning (SSL) scheme for MTN, which enables it to generate semantic-preserving node embeddings for MP4 tree nodes. The design of the SSL scheme helps MTN cope with unseen data in MP4 trees. We also devise a data augmentation technique for MP4 trees, which helps train MTN in data-scarce scenarios.

We evaluate the performance of MTN on the EVA-7K dataset [137], where MTN shows good performance in 3 video forensics analysis tasks. We also show that MTN can gain a more comprehensive understanding on MP4 trees and is more robust to potential attacks compared to existing methods.

5.2 MP4 Tree Network (MTN)

The block diagram of MTN is shown in Figure 5.1. The input MP4 tree is first transformed so that it can be processed by the Graph Neural Networks (GNNs) (Section 5.2.1). Then, the Node Embedding Network (NEN) is used to process the transformed MP4 tree to generate a node embedding for each node in the transformed tree (Section 5.2.2). After that, the Graph Convolution (GC) and Graph Pooling (GP) step analyzes all node embeddings and produce a graph embedding (Section 5.2.3), which is a single vector representation of the MP4 tree. Finally, the graph embedding vector is used by a classifier in the classification step to predict a label for the input MP4 tree depending on the specific forensic analysis task.

5.2.1 MP4 Tree Transformation

MP4 video files are organized using a tree data structure [134], [135] (as shown in Figure 5.2). An MP4 video file is made up of a set of nodes, where each node contains the following information:

- The node type (*e.g.* `ftyp`, `moov`), which annotates the type of data stored in the node and its descendants.
- The node data, which carries the information in the node. Pieces of information are stored as a list of key-value pairs (*e.g.* (key_1, val_1) , (key_2, val_2)). Each key is a string, while each value can be a string, number, list, or dictionary.
- The list of child nodes, which is used to maintain the tree structure.

Note that path-like strings can be used to point at nodes and node data in an MP4 tree. For example, `moov/udta` points to the `udta` node shown in Figure 5.2. The string `moov/udta/@key2=val2` points to a key-value pair stored in the `udta` node. Here, `@` and `=` are the prefix and delimiter for key-value pairs, respectively.

Analyzing an MP4 tree is far from being a simple task, as the node data contains variable amount of key-value pairs, and the value of a pair can be another data structure such as list or dictionary. MP4 trees have to be transformed so that they can be processed by GNNs. The MP4 tree transformation process we propose takes as input an MP4 tree \mathcal{T} and returns a transformed version of the tree \mathcal{T}_t . In \mathcal{T}_t , each node contains the following information:

- The tag, which indicates the type of node.
- The data, which is either a String Object List (SOL) or a number.
- The list of child nodes.

The MP4 tree transformation is a two-step process. In the first step, the MP4 tree \mathcal{T} is restructured into a new tree \mathcal{T}_r . In the second step, the strings in \mathcal{T}_r are processed to generate the transformed tree \mathcal{T}_t . We provide more details about these two steps.

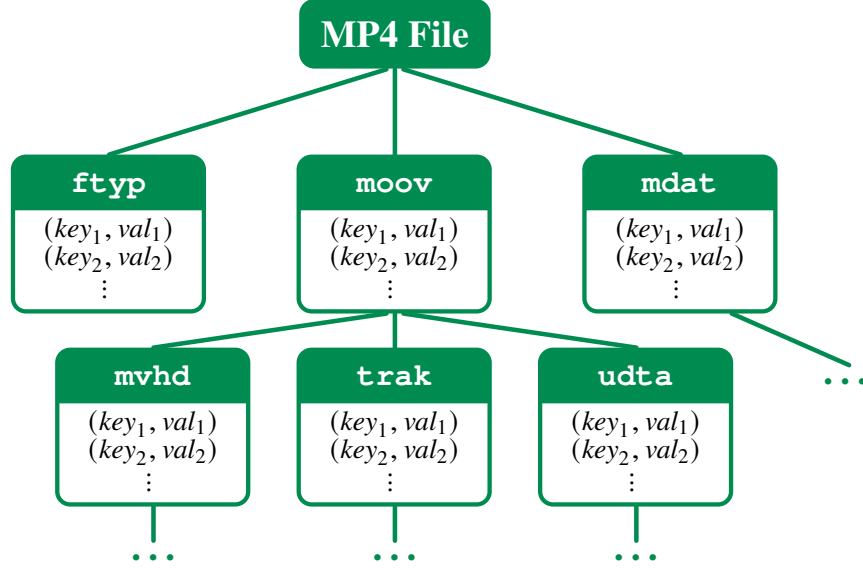


Figure 5.2. The illustration of the tree structure of MP4 video files.

5.2.1.1 MP4 Tree Restructuring

This step maps complicated data structures (*e.g.* list and dictionary) in an MP4 tree \mathcal{T} to graph information that can be processed by GNNs. It makes sure that every node in the resulting tree \mathcal{T}_r contains one string or one number as data.

In tree restructuring, we represent complicated data structures in the \mathcal{T} using a series of nodes and connections in \mathcal{T}_r . For example, a key-value pair (key, val) in \mathcal{T} can be represented by two interconnected nodes in \mathcal{T}_r , where the first node has tag k and data key , and the second node has tag v and data val . A list in \mathcal{T} can be represented by creating a new node ℓ' with tag l in \mathcal{T}_r as the “header” of the list, and for each item in the list, create a child node of ℓ' with tag v that stores the item as its data. A precise definition of the tree restructuring procedure can be found in Section 5.6.1. In the restructured tree \mathcal{T}_r , each node is associated with one of the four tags k , v , l , or n . k nodes and v nodes are used to represent keys and values, respectively. A key-value pair in the original tree is represented as an edge between a k node and a v node. l nodes are list headers, whose child nodes are items in the list. n nodes represent nodes in the original MP4 tree as well as dictionary headers, whose child nodes are k nodes or n nodes. If all values in the MP4 tree

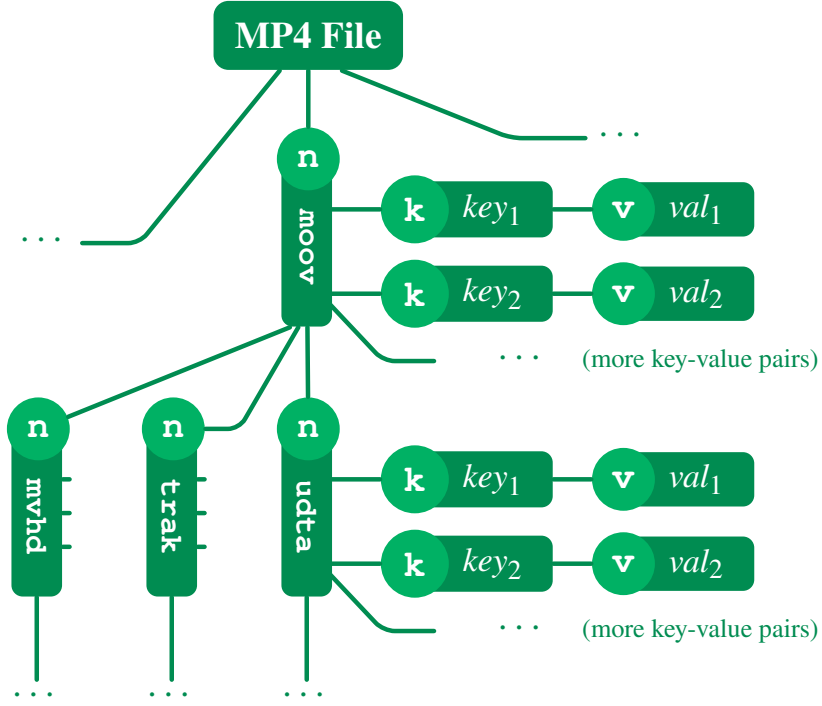


Figure 5.3. The restructured MP4 tree derived from the tree illustrated in Figure 5.2. For each node, the tag is shown in the circle and the data is shown in the rectangle.

shown in Figure 5.2 are strings or numbers, then the corresponding restructured MP4 tree is shown in Figure 5.3.

The restructured tree preserves almost all information in the original tree (only the order of list items is missing) while ensuring that the data of each node is either a string or a number. This makes it easier for GNNs to analyze the restructured MP4 trees.

5.2.1.2 String Processing

After the tree restructuring, the data of each node in \mathcal{T}_r is either a string or a number. A string in \mathcal{T}_r can be further divided to multiple parts that contain heterogeneous information. For example, the string `Lavf58.29.100` indicates that the MP4 file contains the signature from the `libavformat` encoding library in `ffmpeg`⁵ version 58.29.100. It is ideal for the analysis method to see this string represented as `Lavf, 58, 29, 100` where `Lavf` is a string and `58, 29, 100` are numbers. The string processing step removes noise, partitions the string into “words”, and

⁵<https://ffmpeg.org/>

distinguishes between textual data and numerical data. A detailed description of this step can be obtained from Section 5.6.2.

This step converts each string in \mathcal{T}_r into a list of objects known as SOLs, where each object in the SOL is either a textual word or a real number. The resulting tree is the transformed MP4 tree, which is denoted by \mathcal{T}_t . After this step, the data of each node in the transformed tree \mathcal{T}_t is either an SOL or a number. SOLs are lists, but they do not make the structure of \mathcal{T}_t more complicated. As described in Section 5.2.2, the information from all items in an SOL will be summarized into a single fixed-size vector.

5.2.2 Node Embedding Network (NEN)

Graph Neural Networks (GNNs) are neural networks that can analyze graph data structures [144], [145]. Since the transformed MP4 trees are a special case of generic graphs, we can use GNNs to analyze them.

The proposed Node Embedding Network (NEN) is based on GNNs. The NEN analyzes a transformed MP4 tree \mathcal{T}_t and produce a node embedding for each node in \mathcal{T}_t . The block diagram of NEN is shown in Figure 5.4. NEN is made up of three components: the number encoder, the SOL encoder, and the Graph Analysis Module (GAM).

The number encoder (Section 5.2.2.1) generates a vector representation for numbers in \mathcal{T}_t . The SOL encoder (Section 5.2.2.2) generates a vector for an SOL that summarizes information from all items in the SOL. Using the number encoder or SOL encoder, the data from the nodes in \mathcal{T}_t are represented using real-valued vectors known as node data vectors. The node data vectors are processed by the GAM (Section 5.2.2.3) to produce the node embeddings, which are M -dimensional real-valued vectors. In Section 5.2.2.4, we describe how the NEN can be pretrained using Self-Supervised Learning (SSL) techniques so that the node embeddings contain semantic information about the MP4 tree.

Note that Natural Language Processing (NLP) approaches [146] are used in the NEN to process numbers and SOLs. The word embedding technique proposed in [127], [147] is used to convert words into word embeddings, which are M -dimensional vectors that can be updated during training. In the following, we use the arc symbol over a word ($\overline{\text{word}}$) to denote the embedding of the word.

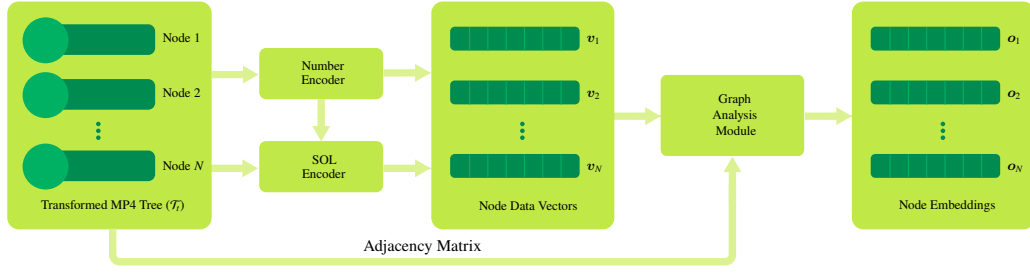


Figure 5.4. The block diagram of NEN.

5.2.2.1 Number Encoder

The role of the number encoder is to represent numbers as M -dimensional vectors. Although it is possible to use existing NLP approaches such as BERT [147] for this conversion, it has been shown in [148] that the number representations generated by most existing NLP approaches can be inefficient and can lead to worse performance in number-related reasoning tasks. Therefore, in the NEN, we explicitly represent numbers as the linear combination of two known vectors. This representation is more efficient since it only requires two M -dimensional vectors. The represented number can also be easily retrieved once the two vectors are given.

The number encoder operation is described as follows. It first converts the input number to symmetric log scale, which is defined as

$$\text{symlog}(x) = \text{sign}(x) \cdot \log_{10} (1 + |x \cdot \ln(10)|) . \quad (5.1)$$

The image of $\text{symlog}(x)$ is shown in Figure 5.5. The value of $\text{symlog}(x)$ preserves the sign and value of x while compressing the scale of x .

Define the clip operation as

$$\text{clip}(x, a, b) = \max [a, \min(x, b)] , \quad (5.2)$$

where a is the lower bound and b is the upper bound. From the data, we determine the minimum and maximum numbers to be represented by the number encoder, which are denoted by τ_{\min} and

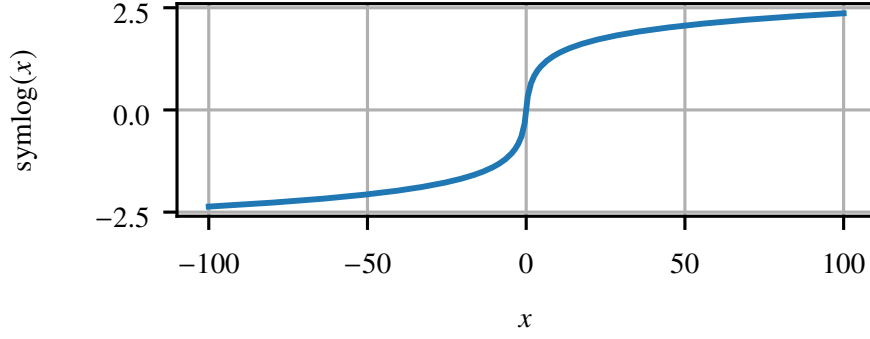


Figure 5.5. The image of $\text{symlog}(x)$.

τ_{\max} , respectively. We add two special words $\#NUM1$ and $\#NUM2$ to the set of all words. Their corresponding word embeddings $\overline{\#NUM1}$ and $\overline{\#NUM2}$ are the vector representations of τ_{\min} and τ_{\max} . All numbers between τ_{\min} and τ_{\max} can be represented as a linear combination of $\overline{\#NUM1}$ and $\overline{\#NUM2}$.

Some numbers (e.g., UNIX timestamps) in the MP4 tree are used to represent time stamps. To preserve the semantic meaning of such numbers and separate them from “plain” numbers, we introduce a time projection matrix $P_{\text{time}} \in \mathbb{R}^{M \times M}$, which can be updated during training. As a result, the number encoder operation, denoted by $\text{Nenc}(\cdot)$, can be written as

$$\text{Nenc}(x) = P \left(s(x) \overline{\#NUM1} + [1 - s(x)] \overline{\#NUM2} \right), \quad (5.3)$$

where

$$s(x) = \frac{\text{symlog}[\text{clip}(x, \tau_{\min}, \tau_{\max})] - \text{symlog}(\tau_{\min})}{\text{symlog}(\tau_{\max}) - \text{symlog}(\tau_{\min})}, \quad (5.4)$$

and

$$P = \begin{cases} \text{identity} & \text{if } x \text{ is a plain number,} \\ P_{\text{time}} & \text{if } x \text{ is a time stamp number.} \end{cases} \quad (5.5)$$

It can be seen that $\text{Nenc}(x) \in \mathbb{R}^M$.

5.2.2.2 SOL Encoder

As described in Section 5.2.1.2, a transformed MP4 tree contains SOLs. The role of the SOL encoder is to map each SOL into an M -dimensional vector.

Items in an SOL are either textual words or numbers. Denote an SOL of length L by $\{\rho_1, \dots, \rho_L\}$. We first convert each item in the SOL into a vector representation using the following rule:

$$\rho'_i = \begin{cases} \widehat{\rho_i} + e_i & \text{if } \rho_i \text{ is a textual word,} \\ \text{Nenc}(\rho_i) + e_i & \text{if } \rho_i \text{ is a number,} \end{cases} \quad (5.6)$$

where $\rho'_i \in \mathbb{R}^M$, and $e_i \in \mathbb{R}^M$ is the positional embedding [69] for the i -th item in the SOL that can be updated during training. They added to the vector representations of the SOL items to encode the order of items. We also encode the length of the SOL by computing $\rho'_0 = \text{Nenc}(L) + e_0$.

We use the transformer encoder [66] to analyze the sequence of vectors $\{\rho'_0, \rho'_1, \dots, \rho'_L\}$. The transformer encoder, which is based on the Multihead Self Attention (MSA) mechanism, has been successful in many sequence processing tasks [69], [147], [149]. The output of the transformer encoder is a new sequence of vectors $\{\tilde{\rho}'_0, \tilde{\rho}'_1, \dots, \tilde{\rho}'_L\}$, $\tilde{\rho}'_i \in \mathbb{R}^M$. The output of the SOL encoder is the average of the transformer encoder output. That is, the SOL encoder operation (denoted by $\text{Senc}(\cdot)$) can be written as

$$\text{Senc}(\{\rho_1, \dots, \rho_L\}) = \frac{1}{L+1} \sum_{i=0}^L \tilde{\rho}'_i. \quad (5.7)$$

After the SOL encoder operation, the information in the SOL is summarized in an M -dimensional vector.

5.2.2.3 Graph Analysis Module (GAM)

The Graph Analysis Module (GAM) analyzes the tag, data, and parent/child nodes of each node in the transformed MP4 tree. It consists of two components: node tag encoding and Graph Attention Network (GAtN). In this section, we assume there are N nodes in the transformed MP4 tree \mathcal{T}_t . The tag of each node is denoted by t_1, \dots, t_N , where $t_i \in \{\text{k}, \text{v}, \text{l}, \text{n}\}$. The data of each node is denoted by

d_1, \dots, d_N , where d_i is either an SOL or a number. The node data vector of the i -th node, denoted by v_i , is given by

$$v_i = \begin{cases} \text{Senc}(d_i) & \text{if } d_i \text{ is an SOL,} \\ \text{Nenc}(d_i) & \text{if } d_i \text{ is a number.} \end{cases} \quad (5.8)$$

5.2.2.3.1 Node tag encoding.

The node data vectors v_i contain information about the node data. To encode the tag information of the nodes, we use the linear projection approach described in [145]. More concretely, we introduce four $M \times M$ matrices P_k , P_v , P_n , and P_l , which can be updated during training. We use them to generate the tag encoded node data vector u_i as follows:

$$u_i = P_{t_i} v_i. \quad (5.9)$$

The vectors u_i are used as the input to the GAtN step.

5.2.2.3.2 GAtN

GAtNs are based on the Graph Attention (GA) [144] technique, which is an adaptation to the Multihead Self Attention (MSA) mechanism [66] so that graph-structured data can be processed. MSA is based on Self Attention (SA) mechanism. The MSA introduces two parameters h and \hbar , which will be described later. Suppose the input to SA are N vectors w_1, \dots, w_N , where $w_i \in \mathbb{R}^M$. SA introduces three $\hbar \times M$ matrices U_1 , U_2 , and U_3 to linearly project w_i 's into three different versions. These three matrices can be updated in training. Define α_{ij} by

$$\alpha_{ij} = \frac{\exp(\eta_{ij})}{\sum_{k=1}^N \exp(\eta_{ik})}, \quad (5.10)$$

where

$$\eta_{jk} = (U_1 w_j)^T (U_2 w_k). \quad (5.11)$$

The SA of the i -th input vector \mathbf{w}_i is given by

$$\text{SA}(\mathbf{w}_i) = \sum_{j=1}^N \alpha_{ij} \mathbf{U}_3 \mathbf{w}_j. \quad (5.12)$$

It can be seen that $\text{SA}(\mathbf{w}_i) \in \mathbb{R}^{\hat{h}}$. In MSA, h different SA results (called “attention heads”, or “heads” for short) are computed simultaneously and combined to allow a more comprehensive analysis of the input [66]. To merge the SA results, a fourth projection matrix $\mathbf{U}_4 \in \mathbb{R}^{M \times M}$ is introduced, which can be updated in training. The MSA operation is given by

$$\text{MSA}(\mathbf{w}_i) = \left\{ \left[\text{SA}_1(\mathbf{w}_i)^T \mid \cdots \mid \text{SA}_h(\mathbf{w}_i)^T \right] \mathbf{U}_4 \right\}^T. \quad (5.13)$$

To ensure that the shape of matrices are valid for multiplication, $h\hat{h} = M$ must hold. Therefore, the number of heads h must divide M , and \hat{h} is set to be M/h . In this case, $\text{MSA}(\mathbf{w}_i)$ has the same dimensionality as \mathbf{w}_i . In the transformer encoder [66], the MSA mechanism is used repeatedly with nonlinear activation to analyze input vector sequence.

GA incorporates the graph connectivity information in MSA so that the attention mechanism can be used to analyze graph data structure. The input to GA is a graph of N nodes and the adjacency matrix. Suppose the data of the N nodes are represented by N vectors $\mathbf{w}_1, \dots, \mathbf{w}_N$, where $\mathbf{w}_i \in \mathbb{R}^M$. For the i -th node in the graph, we can determine the “neighborhood” of the node (denoted by \mathcal{N}_i) based on the adjacency matrix. In our implementation, the neighborhood is defined to the node itself, its parent node, and its child nodes. In GA, Equation (5.10) is modified to

$$\alpha_{ij} = \frac{\exp(\eta_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(\eta_{ik})}, \quad (5.14)$$

and Equation (5.12) is modified to

$$\text{GA}(\mathbf{w}_i) = \sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{U}_3 \mathbf{w}_j. \quad (5.15)$$

That is, for the i -th node in the graph, GA is only computing attention with respect to the nodes in its neighborhood.

5.2.2.4 Self-Supervised NEN Pretraining

SSL refers to training machine learning models without data labels [150]. SSL techniques can obtain information and representation from unlabeled data, which can be helpful for a variety of downstream tasks. It has been used in NLP [147], image processing [151], and audio processing [149]. SSL can reduce the need of training data [149], increase model robustness [150], create semantically meaningful representations [151], and facilitate transfer learning [152]. We pretrain the NEN using SSL to generate node embeddings $\mathbf{o}_1, \dots, \mathbf{o}_N$, where $\mathbf{o}_i \in \mathbb{R}^M$ preserves semantic information about the i -th node. The pretrained NEN allows easier fine-tuning for various video forensics tasks.

To pretrain NEN, we devise three classification and two regression SSL “pretext” tasks. To enable SSL, we introduce a special word $\# \text{MASK}$, whose word embedding $\overline{\# \text{MASK}}$ will be used to mask the node data vectors passed to the GAM. The five pretext tasks are described as below.

1. Masked node data type classification ($(\overline{\text{DC}})$): randomly select from the tree the i -th node and mask its data vector \mathbf{u}_i with $\overline{\# \text{MASK}}$. Use the node embedding \mathbf{o}_i to classify if the data of the i -th node is SOL or number.
2. Masked node tag classification ($(\overline{\text{TC}})$): randomly select from the tree the i -th node and mask its data vector \mathbf{u}_i with $\overline{\# \text{MASK}}$. Use the node embedding \mathbf{o}_i to classify the node tag (k, v, l, n).
3. Masked SOL word inclusion classification ($(\overline{\text{IC}})$): randomly select from the tree the i -th node whose data is SOL and mask its data vector \mathbf{u}_i with $\overline{\# \text{MASK}}$. Then, select two words from the set of all words, where the first word is in the SOL of the i -th node and the second word is not. Classify if the two words are in the SOL of the i -th node using the node embedding \mathbf{o}_i .
4. Masked number regression ($(\overline{\text{NR}})$): randomly select from the tree the i -th node whose data is number and mask its data vector \mathbf{u}_i with $\overline{\# \text{MASK}}$. Regress the symmetric log value of the number of the i -th node using the node embedding \mathbf{o}_i .
5. Masked SOL length regression ($(\overline{\text{LR}})$): randomly select from the tree the i -th node whose data is SOL and mask its data vector \mathbf{u}_i with $\overline{\# \text{MASK}}$. Regress the length of the SOL using the node embedding \mathbf{o}_i .

The $\overline{\text{DC}}\overline{\text{TC}}\overline{\text{NR}}$ tasks examine if the NEN can gain understanding about the MP4 tree by asking it to fill in missing information in the node data vectors. The $\overline{\text{IC}}\overline{\text{LR}}$ tasks examine if the node embeddings of SOL data preserve the information in the SOL.

The set of words and their word embeddings determined at training time may not include all words that can appear in an MP4 tree. To cope with unseen words, we introduce a special word $\overline{\text{\#UNKNOWN}}$. When a word in the MP4 tree has not been seen, we will use $\overline{\text{\#UNKNOWN}}$ as its word embedding. To emulate the presence of unseen words during training, in the NEN pretraining, we randomly select 5% of the words in the transformed tree and replace their word embeddings with $\overline{\text{\#UNKNOWN}}$.

5.2.3 Graph Convolution and Graph Pooling

In popular Convolutional Neural Network (CNN) based image classification techniques such as VGG [153] and ResNet [154], convolution layers are used to extract features from the input image and pooling layers are used to downsample the extracted features so that subsequent layers can focus on higher level features. Similar approach can be used for graphs using Graph Convolution (GC) and Graph Pooling (GP) techniques. The node embeddings generated by the NEN trained with SSL contains semantic information about the nodes in the MP4 tree. Processing the node embeddings with GC and GP techniques allows the neural network to focus on node data and graph topology that have significant contribution to the decision. In MTN, we follow the GC technique proposed by Kipf *et al.* [155] and the GP technique proposed by Lee *et al.* [156].

In GC, the node embedding is first stored in a matrix $\mathbf{E} \in \mathbb{R}^{N \times M}$, where N is the number of nodes and M is the dimensionality. The GC operation is defined as

$$\text{GC}(\mathbf{E}) = \text{ReLU}\left(\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{E}\boldsymbol{\theta}\right), \quad (5.16)$$

where $\mathbf{D} \in \mathbb{R}^{N \times N}$ is the degree matrix, $\mathbf{A} \in \mathbb{R}^{N \times N}$ is the adjacency matrix, and $\boldsymbol{\theta} \in \mathbb{R}^{M \times C}$ is the parameters of GC which can be updated during training. Here, C is a hyperparameter that determines the number of “channels” of GC. It can be seen that $\text{GC}(\mathbf{E}) \in \mathbb{R}^{N \times C}$.

In GP, the graph is “downsampled” by a factor of $\alpha \in (0, 1)$. At first, a graph SA score $\mathbf{Z} \in \mathbb{R}^N$ is computed by

$$\mathbf{Z} = \tanh \left(\mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} \text{GC}(\mathbf{E}) \boldsymbol{\theta}_{\text{att}} \right), \quad (5.17)$$

where $\boldsymbol{\theta}_{\text{att}} \in \mathbb{R}^C$ is the parameters of GP which can be updated during training. Then, the nodes associated with the top $\lceil \alpha N \rceil$ scores in $\mathbf{Z} \in \mathbb{R}^N$ are preserved in the graph, while the remaining nodes are removed from the graph. That is, the output of GP (denoted by \mathbf{E}') is a $\lceil \alpha N \rceil \times C$ matrix, whose rows are from those in $\text{GC}(\mathbf{E})$ associated with top scores. After node removal, the degree matrix \mathbf{D} and adjacency matrix \mathbf{A} need to be updated accordingly.

5.3 Data Augmentation

Video data files are usually significantly larger compared to image, audio, and text data. Therefore, it is difficult to create, transmit, and store large video datasets. In practice, we need to analyze video datasets that are several gigabytes in size, but only contain several hundred samples per label. The lack of data can be challenging for the training of deep neural network models. To cope with this problem, we devise a data augmentation technique for training MTN.

The proposed data augmentation technique is derived from decision tree analysis for MP4 trees [137]. In this class of methods, the nodes and key-value pairs in an MP4 tree are first converted into path-like strings (as described in Section 5.2.1). Then, the occurrence of each unique string is counted throughout the tree, which effectively creates an integer-valued vector representation (denoted by \mathbf{q}) for the MP4 tree. The vector \mathbf{q} is used by decision tree classifiers to predict labels for the MP4 tree. To classify an MP4 tree based on \mathbf{q} , the decision tree must have split on a number of elements in \mathbf{q} , where each split either indicates the presence of a string or the absence of a string. We focus on the splits that correspond to the presence of a string and denote such strings by s_1, \dots, s_R . Each path-like string s_i represents a path in the MP4 tree from the root node to an inner node or a leaf node that contributed to the classification of the MP4 tree. Therefore, we mark all nodes in the MP4 tree along the path as *visited*. When the node marking procedure is done for strings s_1, \dots, s_R , the nodes in the MP4 tree not marked as visited are less likely to have contribution to the decision process, which implies that they can be removed from the tree.

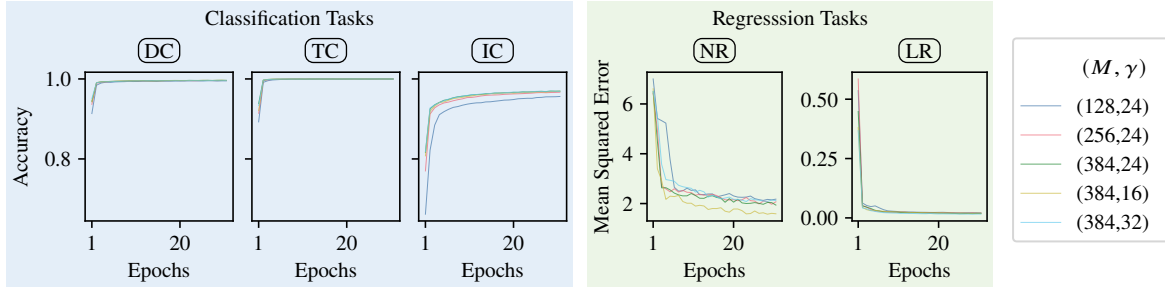


Figure 5.6. The training performance of pretext tasks using different (M, γ) settings.

The splits from one decision tree may overfit the dataset and be subject to the noise in the data. To mitigate this issue, the proposed data augmentation technique is based on a random forest [157] trained using q . Random forests are ensemble classifiers that aggregates the decisions from many decision trees (in our experiments, we used 100 decision trees). For each decision tree in the random forest, if it classifies the label correctly, we collect the strings s_i as described above and add the strings to the set of all strings. The node marking procedure is used for the set of all strings. For the nodes that are not marked as visited, we randomly remove β percent of them as the data augmentation step. In our experiments, we set $\beta = 30$. The data augmentation is not enabled when the random forest cannot classify the MP4 tree correctly, as the evidence used by the random forest can be unreliable.

5.4 Experiments and Results

In this section, we describe the data, experiments, and results. Throughout the experiments, we extracted the MP4 tree information from MP4 files using the technique described in [94]. We used F₁-score [158] and balanced accuracy [84] to measure the performance of various approaches.

5.4.1 NEN Pretext Tasks

The NEN was pretrained on the MFC dataset [159], which contains 4038 MP4 video files. In the NEN, the SOL encoder consists of 4 transformer encoder layers; the GAtN consists of 6 GA layers. We set the number of heads in the MSA (*i.e.* h) to be 4. Details about the NEN architecture can be obtained from Section 5.6.5.

We pretrained the NEN using the AdamW optimizer [160] with an initial learning rate of 10^{-5} and a weight decay factor of 0.001. The batch size was set to 16. The classification pretext tasks were trained using the cross entropy loss [161]; the regression pretext tasks were trained using the mean squared error loss [161]. For each batch, the five tasks were trained sequentially. That is, we computed the loss with respect to the first task and updated the model before proceeding to the second task. The training was stopped after 30 epochs.

For the NEN, we need to determine two hyperparameters M (the dimensionality of embeddings) and γ (number of masked nodes in each graph). To determine the choice of parameters, we sampled combinations of (M, γ) and pretrained the NEN. We evaluated the training performance of each task on the MFC dataset after each epoch. The classification tasks were evaluated using accuracy [162] and regression tasks were evaluated using mean squared error [161]. We first fixed $\gamma = 24$ and chose M among 128, 256, and 384. The training performance of NEN is shown in Figure 5.6. It can be seen that using $M = 384$ resulted in the best performance in \mathbb{IC} . Then, we fixed $M = 384$ and chose γ between 16 and 32. The performances are also shown in Figure 5.6. It can be seen that using $\gamma = 16$ resulted in the best performance in \mathbb{NR} . Therefore, in further experiments, we selected $M = 384$ and $\gamma = 16$.

From Figure 5.6, it can also be seen that the performance of all five tasks were generally improving as the number of epochs increased. This shows that the design of the five tasks allows them to function collaboratively to help the NEN learn from unlabeled MP4 trees.

5.4.2 Forensics Analysis Using MTN

For video forensics analysis tasks, the experiments were conducted on the EVA-7K dataset [137]. The dataset consists of 7000 MP4 video sequences captured by different mobile devices. There are 140 pristine video sequences in the dataset, which are the original video files captured by the mobile devices. All 140 pristine video sequences are manipulated using off-the-shelf tools such as *ffmpeg*⁶, *Kdenlive*⁷, and *Adobe Premiere*⁸. This results in 1260 manipulated videos. The pristine

⁶<https://ffmpeg.org/>

⁷<https://kdenlive.org/en/>

⁸<https://www.adobe.com/products/premiere.html>

and manipulate video sequences are uploaded to four social networks (YouTube, Facebook, TikTok, and Weibo) and then downloaded back. In the end, this results in 7000 video sequences.

We evaluated the performance of MTN in 3 video forensics analysis tasks: social network attribution, editing tool attribution, and manipulation detection. In each task, the EVA-7K dataset was divided into training, validation, and testing sets, with a ratio of 4:2:4.

We used the pretrained NEN with $M = 384$ and $\gamma = 16$ (see Section 5.4.1). We used three GC+GP layers with $C = 128$ and $\alpha = 0.5$ (see Section 5.2.3). The classification step of MTN is realized with a three-layer multilayer perceptron network. Details about the MTN architecture can be obtained from Section 5.6.3.

For each training batch, we first sampled 16 MP4 trees from the training set. These 16 samples were added to the training batch. Then, for each sample, if it was classified correctly by the random forest approach (see Section 5.3), we applied data augmentation to the sample twice and put the two output MP4 trees in the training batch. Therefore, the batch size varies between 16 and 48. The MTN was trained using the AdamW optimizer [160] with a learning rate of 3×10^{-5} and a weight decay factor of 0.001. The training was stopped when the validation performance no longer increased. We report the performance of MTN on the testing set. The performance of MTN in each forensics task is reported as below.

- Social network attribution: in this task, the Video Forensics Method (VFM) is given videos downloaded from four social networks as well as local videos that are not from social networks. The VFM is asked to predict the source of the video, which can be either one of the four social networks or local. The performance comparison of this task is shown in Table 5.1.
- Editing tool attribution: in this task, the VFM is given videos that are edited by five video editing tools as well as unedited videos. The VFM is asked to predict which tool is used to edit the video, which can be either one of the five tools or unedited. The performance comparison of this task is shown in Table 5.2.
- Manipulation detection: in this task, the VFM is given videos that are manipulated by video editing tools as well as pristine videos. The VFM is asked to predict whether the video is manipulated. The performance comparison of this task is shown in Table 5.3.

Table 5.1. The F_1 -score comparison of the social network attribution task. The performance of “(Local)” category is not available for Yang *et al.* [137].

Social Network	Yang <i>et al.</i> [137]	Xiang <i>et al.</i> [94]	MTN
YouTube	1.00	0.99	1.00
Facebook	1.00	1.00	1.00
WeiBo	0.99	0.99	0.99
TikTok	1.00	1.00	1.00
(Local)	–	0.99	0.99

Table 5.2. The F_1 -score comparison of the editing tool attribution task.

Tool	Yang <i>et al.</i> [137]	Xiang <i>et al.</i> [94]	MTN
Avidemux	0.99	0.98	0.99
Exiftool	0.98	1.00	0.96
ffmpeg	0.94	1.00	1.00
Kdenlive	0.95	1.00	0.99
Premiere	1.00	0.99	1.00
(Unedited)	0.97	1.00	0.96

Table 5.3. The balanced accuracy score comparison of the manipulation detection task.

Balanced Accuracy	
Güera <i>et al.</i> [136]	0.67
Iuliani <i>et al.</i> [135]	0.85
Yang <i>et al.</i> [137]	0.98
Xiang <i>et al.</i> [94]	0.99
MTN	1.00

It can be seen that MTN achieved the best performance in social network attribution and manipulation detection. The performance of MTN is in line with that of [94]. However, as we will discuss in the next section, MTN has more comprehensive understanding about MP4 trees and is more robust against attacks.

5.4.2.1 The Robustness of MTN

Existing MP4 tree analysis approaches such as [94], [137] use hand-crafted features with decision tree classifiers. Decision tree classifiers function by analyzing a series of deterministic splits on

selected elements in the hand-crafted feature vector. Therefore, this class of methods cannot model the topology of the MP4 tree or node-to-node relationships in the tree, which makes them more vulnerable against attacks. In this section, we demonstrate an evidence removal attack dedicated to methods for MP4 tree analysis based on decision trees, and we show that MTN is more robust against this attack.

In evidence removal attack, the attacker attempts to remove information from the MP4 tree in order to change the predicted label from VFMs. For the attack, we trained a random forest classifier for the social network attribution task that consists of 100 decision trees. For a given sample that is classified correctly by the random forest, we focus on the decision trees in the random forest that also classifies the sample correctly. We gathered all the splits in the focused decision trees that correspond to the presence of a node or key-value pair in the MP4 tree. This effectively generates a set of node and key-value pairs that serves as evidence for the decision. The set of evidence usually contains more than 20 elements. From the set of evidence, we randomly remove 1, 2 . . . , 5 node(s)/key-value pair(s) from the MP4 tree and ask both the random forest and MTN to classify the MP4 tree with evidence removal.

Both the random forest and MTN can output a confidence score for the prediction, which is the probability that the sample belongs to the ground truth label. By comparing the confidence score of the MP4 tree before and after evidence removal, we can evaluate the robustness of the classifiers against the evidence removal attack. The evidence removal attack was applied to all MP4 trees in the testing set, and we computed the average confidence score change caused by the attack.

The result is shown in Table 5.4. It can be seen that the confidence score decline of MTN is less significant compared to the random forest. Note that as the number of evidence removal increases, the confidence score from random forest continues to drop, whereas the confidence score from MTN almost stopped decreasing. This implies that the decision of MTN is likely to be based on a more comprehensive analysis of the MP4 tree, which makes MTN more robust against evidence removal attacks.

Table 5.4. The robustness of random forest and MTN against the evidence removal attack.

Method	Avg. Confidence Before Attack	Confidence Score Change				
		Num. Evidence Removal				
		1	2	3	4	5
Random Forest	0.99	-0.08	-0.17	-0.24	-0.31	-0.38
MTN	0.97	-0.08	-0.14	-0.18	-0.20	-0.21

5.4.3 Ablation Study

We analyzed the effectiveness of NEN pretrain and data augmentation. We trained MTN on the social network attribution task with four settings: with both techniques (+Both); with only data augmentation (-NEN pretrain); with only NEN pretrain (-Data Aug.); with none of the techniques (-Both). We monitored the validation balanced accuracy of the four settings every 3 training steps. The result is shown in Figure 5.7. It can be seen that NEN pretrain and data augmentation combined led to faster training as well as less performance fluctuation. As more training steps elapsed, the MTN trained using both techniques also managed to achieve the best validation balanced accuracy, which was close to 1.

5.5 Conclusion

In this chapter, we propose MTN for video forensics analysis based on end-to-end GNNs. MTN uses the MP4 tree information to make decisions, it does not require any pixel data. We devise an SSL scheme to pretrain MTN, which allows it to generate semantic-preserving node embeddings. We also propose a data augmentation technique for MP4 trees to help train MTN in data-scarce scenarios. The experimental results showed that MTN achieved good performance in video forensics analysis tasks. It is shown that MTN can gain more comprehensive understanding about the MP4 trees and is more robust to the evidence removal attack compared to existing methods. We also demonstrated that the SSL scheme and the data augmentation technique can reduce training iterations, reduce performance fluctuation, and improve the performance. The source code of MTN is available at <https://gitlab.com/viper-purdue/mtn>.

In the future, we will examine the performance of MTN in more video forensics analysis tasks. We will evaluate the scalability of MTN by analyzing large real-world datasets.

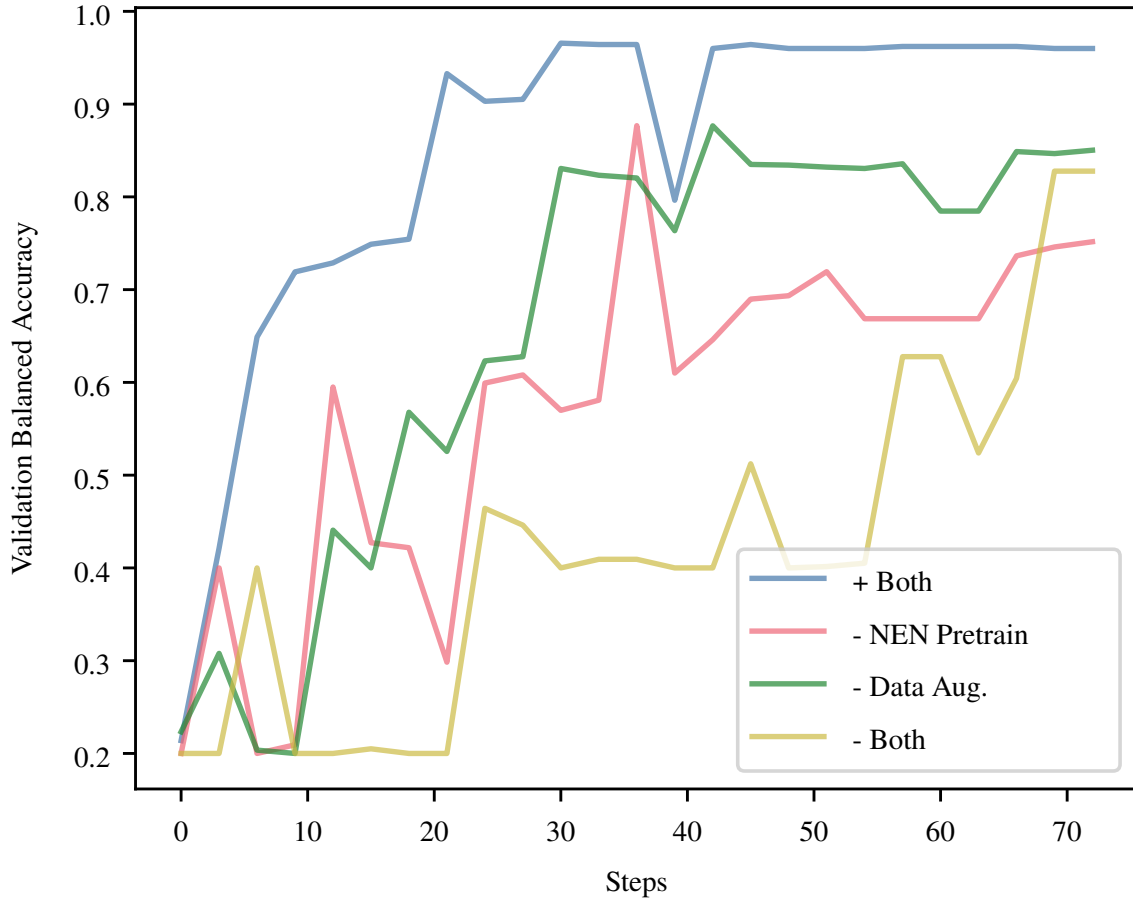


Figure 5.7. The validation performance of different training settings vs. training steps.

5.6 Supplementary Materials

5.6.1 The Tree Restructuring Algorithm

The tree restructuring algorithm is shown in Algorithm 1. The `TreeRestructre` procedure is used on an MP4 tree \mathcal{T} to generate the restructured tree \mathcal{T}_r .

5.6.2 String Processing

The string processing step consists of the following steps:

- Noise removal: non-printable and non-ASCII characters are removed from the input string

Algorithm 1: The tree restructuring procedure.

```

1 Procedure TreeRestructure ( $\mathcal{T}$ ):
   Input: MP4 tree  $\mathcal{T}$ 
   Output: restructured MP4 tree  $\mathcal{T}_r$ 
2   Create a new empty tree  $\mathcal{T}_r$ ;
   // Make sure the new tree  $\mathcal{T}_r$  has the same topology as  $\mathcal{T}$ 
3   foreach node  $n$  in  $\mathcal{T}$  do Create a new node  $n'$  in  $\mathcal{T}_r$  with tag  $n$ ;
4   foreach node  $n$  in  $\mathcal{T}$  do
5       foreach child node  $c$  of  $n$  do
6           Find the corresponding nodes of  $n$  and  $c$  in  $\mathcal{T}_r$ , which are denoted by  $n'$  and  $c'$ ;
7           Connect  $n' \rightarrow c'$  in  $\mathcal{T}_r$ ;
8       end
9   end
   // Add data from  $\mathcal{T}$  to  $\mathcal{T}_r$ 
10  foreach node  $n$  in  $\mathcal{T}$  do
11      Find the corresponding node of  $n$  in  $\mathcal{T}_r$ , which is denoted by  $n'$ ;
12      foreach key-value pair  $(key, val)$  in  $n$  do RecursiveAdd  $((key, val), n')$  ;
13  end
14  return  $\mathcal{T}_r$ 

15 Procedure RecursiveAdd ( $obj, u'$ ):
   Input: an object to be added ( $obj$ ), which can be a string, a number, a key-value pair, a list, a dictionary;
   the node where the data is added ( $u'$ )
   // Recursively adds data into the restructured tree  $\mathcal{T}_r$ 
16  if  $obj$  is a number or a string then
17      Create a new node  $v'$  in  $\mathcal{T}_r$  with tag  $v$ ;
18      Set the data of  $v'$  as  $obj$ ;
19      Connect  $u' \rightarrow v'$  in  $\mathcal{T}_r$ ;
20  else if  $obj$  is a key-value pair  $(key, val)$  then
21      Create a new node  $k'$  in  $\mathcal{T}_r$  with tag  $k$ ;
22      Set the data of  $k'$  as  $key$ ;
23      Connect  $u' \rightarrow k'$  in  $\mathcal{T}_r$ ;
24      RecursiveAdd ( $val, k'$ )
25  else if  $obj$  is a dictionary then
26      Create a new node  $w'$  in  $\mathcal{T}_r$  with tag  $n$ ;
27      Set the data of  $w'$  as #DICT;
28      Connect  $u' \rightarrow w'$  in  $\mathcal{T}_r$ ;
29      foreach  $(key, val)$  in  $obj$  do RecursiveAdd  $((key, val), w')$  ;
30  else if  $obj$  is a list then
31      Create a new node  $w'$  in  $\mathcal{T}_r$  with tag  $l$ ;
32      Set the data of  $w'$  as #LIST;
33      Connect  $u' \rightarrow w'$  in  $\mathcal{T}_r$ ;
34      foreach item in  $obj$  do RecursiveAdd ( $item, w'$ ) ;

```

- String partition: the input string is broken down into parts separated by white space characters, where each part can be a textual word, URL, version string, ratio string, or numeric string
- URL decomposition: the input strings can also contain URLs which are decomposed into multiple parts; for example, `www.example.com/about/index.html` will be broken down into `www.example.com`, `about`, `index.html`
- Version string decomposition: version strings found in the input string are decomposed into multiple numeric parts; the numeric parts are surrounded by two special words `#VER_BEG` and `#VER_END` to emphasize that they are the members of a version string
- Ratio string conversion: ratio strings such as `1/48000` found in the input string are converted into their corresponding numeric values
- Numeric string conversion: string representations of floating point numbers and integers found in the input string are converted into their corresponding numeric values

5.6.3 Graph Pooling (GP) Readout

Denote the output of the GP layer by E' . The first dimension of E' is variable. To produce a fixed-sized vector for classification, a readout mechanism [156] is defined as follows:

$$\text{ReadOut}(E') = \text{MaxP}(E') || \text{MinP}(E'). \quad (5.18)$$

Here, MaxP denotes the max pooling operation that determines the maximum element along each column of E' , MinP denotes the min pooling operation, and $||$ denotes concatenation. The readout operation always produces a $2C$ -dimensional vector, which can be used by a classifier. Here, C is the number of “channels” in the GC layer.

In MTN, we use the hierarchical pooling architecture described in [156]. That is, there are multiple GC+GP layers in the GNN. The readout operation is used for each GP layer in the GNN, and the results are summed together to generate a single vector for classification.

5.6.4 Model Size of the Node Embedding Network (NEN)

The relationship between the choice of M (embedding size) and the number of parameters in the NEN are shown in Table 5.5.

Table 5.5. The choice of M and the corresponding size of the NEN.

M	128	256	384
Num. Parameters	6.56M	15.39M	26.52M

5.6.5 Model Architecture

We implemented MTN using the `PyTorch` library⁹. We used the GC and GP implementation from the `PyG` library¹⁰.

We used `PyTorch`’s transformer encoder implementation (`TransformerEncoderLayer`). In the number encoder, we chose $\tau_{\max} = 10^{12}$ and $\tau_{\min} = -\tau_{\max}$ (see Sec. 2.2.1). The SOL encoder in the NEN contains 4 transformer encoder layers with $M = 384$ and $h = 4$. The GAtN in the GAM contains 6 transformer encoder layers with $M = 384$ and $h = 4$. The choice of M is discussed in Sec. 4.1. For GC, we used the `GCNConv` layer from the `PyG` library with $C = 128$. For GP, we used the `SAGPool` layer from the `PyG` library with $\alpha = 0.5$. The definition of C and α can be found in Sec. 2.3. The MTN has 3 GC+GP layers. Under this setting, the GC+GP layers contain 281k parameters. For classification, we used a three layer multilayer perceptron network with $2C, 2C, K$ neurons. Here, K is the number of classes in the video forensics task.

⁹<https://pytorch.org/docs/stable/index.html>

¹⁰<https://pytorch-geometric.readthedocs.io/en/latest/>

6. EXTRACTING EFFICIENT SPECTROGRAMS FROM MP3 COMPRESSED SPEECH SIGNALS FOR SYNTHETIC SPEECH DETECTION

6.1 Introduction

Speech compression can greatly reduce the data rate of signals while maintaining quality. Therefore, speech compression techniques such as MP3 [163] and AAC [164] are widely used in transmission and storage applications. The use of compression allows new approaches for forensics analysis [139], [165]. For example, the MP3 compression artifacts can be used for audio splicing localization [166], [167]. In this paper, we demonstrate that it is possible to approximate the spectrogram [168] efficiently for a MP3 compressed signal. We denote the spectrograms extracted using our approach by Efficient Spectrograms (E-Specs). E-Spec can be used to significantly reduce the complexity of spectrogram computation and the time required for MP3 decoding. E-Spec does not contain the artifacts created by the MP3 synthesis filterbank, which makes it useful in audio forensics scenarios. E-Spec can be formulated as a fast approximation to the Short Time Fourier Transform (STFT), which is the foundation for many other signal features such as Mel-Frequency Cepstrum Coefficients (MFCC) [169] and Constant-Q Transform (CQT) [170]. These speech features are used extensively in speech analysis tasks including speaker recognition [171], speech emotion classification [172], and synthetic speech detection [173]. Note that speech features are generally a subset of audio features. They can be used for other audio analysis tasks such as music genre classification [174], respiratory disease diagnosis [175], and audio style transfer [176].

The main contributions of this paper are:

- We proposed an approach to efficiently compute spectrograms from MP3 compressed signals (denoted by E-Spec); this approach can reduce the computational complexity by ~77.60 percentage points (p.p.) and save ~37.87 p.p. of MP3 decoding time
- We used E-Spec to train synthetic speech detectors, which resulted in the best performance for 3 out of 4 neural network architectures; E-Spec also had the best overall performance

The paper is organized as follows. In Section 6.2, we present existing work. In Section 6.3, we briefly introduce related background. In Section 6.4, we describe the computation of E-Spec. In

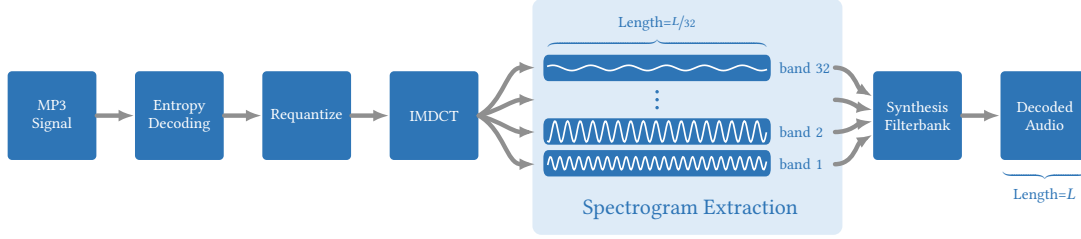


Figure 6.1. The block diagram of the MP3 signal decoding process, which helps illustrate the our efficient spectrogram extraction method. MP3 compression splits the signal into 32 frequency bands using a band-pass analysis filterbank. Extracting the spectrogram from the frequency bands is more efficient because the high frequency bands are redundant; the length of each band is shorter; and the synthesis filterbank is not involved.

Section 6.5, we show the synthetic speech detection performance of E-Spec. In Section 6.6, we conclude the paper and describe future work.

6.2 Related Work

Synthetic speech detection. Synthetic speech detection refers to the detection of synthesized speech. There have been multiple popular synthetic speech datasets/challenges such as ASVspoof19 [173], ASVspoof21 [177], and FoR [178]. Review papers such as [139], [179] provide more comprehensive overview of this topic. Typical methods use deep neural networks with 2D audio features such as the spectrogram [180], MFCC [181], and CQT [182].

MP3 bitstream analysis. MP3 bitstream refers to the analysis of the MP bitstream used for the MP3 decoding process, instead of the fully decoded time domain signal. Existing MP3 bitstream analysis mainly focus on using MP3 encoding parameters such as Modified Discrete Cosine Transform (MDCT) coefficients, scale factors, or Huffman table indices for multiple MP3 compression detection or localization [54], [63], [166], [167]. To our knowledge, our work is the first to use the MP3 bitstream to accelerate the computation of speech features such as the spectrogram.

6.3 Background

In this section, we introduce the background related to the topics discussed.

6.3.1 MP3 Compression

The details of MP3 compression are described in the ISO standard [163] and various materials [73], [74], [183]. To encode a time-domain signal using MP3, the signal is first partitioned into frames of 1152 samples. Each frame is processed by the MP3 encoder separately to produce an MP3 bitstream for that frame. Within each frame, the time domain samples are separated into two granules, where each granule contains half of the 1152 samples (i.e., 576 samples). Granules are the smallest element of MP3 compression. The time domain samples in each granule will be processed by a pseudo Quadrature Mirror Filter (QMF) analysis filterbank (see Section 6.3.2). After filtering, the analysis filterbank generates 32 bands of time-domain signals where each band contains the same number of samples as the time-domain signal. The signal in the i -th band contains the frequency components extracted by H_i . The total number of samples in all bands is 32-fold of the original signal. To maintain the same number of samples, each band is decimated by a factor of 32. In each band, the time domain samples are coded using the MDCT [184]. Finally, all 32 bands will be quantized and entropy coded. The decoding process (Figure 6.1) is almost the same as the encoding process but inverted. The MP3 bitstream is binary decoded to acquire the MDCT coefficients. The reconstruction of the MDCT coefficients is sometimes known as requantization. The Inverse Modified Discrete Cosine Transform (IMDCT) is used to compute the time-domain signals in each band. Then, the time-domain signal in each band is interpolated by a factor of 32. A synthesis filterbank is used to combine the signals from 32 bands and generate the decoded time domain audio signal. Note that the decoded signal has compression artifact in it due the quantization and synthesis processes.

6.3.2 Quadratic Mirror Filter (QMF)

Quadrature Mirror Filter (QMF) [185]–[188] is a type of band-pass filterbank that is used in MPEG 1/2/4 audio coding standards. In MP3, the QMF is used to split the time-domain audio signal into 32 frequency bands. QMF filterbank is a type of modulated filterbank, where the filter for each subband is the modulated version of the baseband low-pass filter. More specifically, denote

the baseband filter by $H[n]$ with length L . The filter for the k -th subband is cosine modulated and is given by

$$H_k[n] = H[n] \cos \cdot \left(\frac{\pi}{N} \cdot \left(k + \frac{1}{2} \right) \cdot \left(n + \frac{1 - LN}{2} \right) + (-1)^{k+1} \cdot \frac{\pi}{4} \right). \quad (6.1)$$

That is, each subband filter is frequency shifted from the baseband filter. As shown in Figure 6.3, the QMF filterbank used in MP3 is made up of 32 subbands H_1, \dots, H_{32} , where H_2, \dots, H_{32} are frequency shifted versions of H_1 .

Given the subband signals, the QMF filterbank can only achieve near perfect reconstruction, which indicates the original signal before applying the filterbank cannot be perfectly retrieved. The reconstructed audio signal will be tampered by artifacts. QMF filterbank also has critical sampling property: the total number of valid samples in all subband signals are the same as that in the original signal. Therefore, after applying K filters in the filterbank to the input audio signal, one can decimate each subband signal by a factor of K , and the remaining samples are enough to achieve near perfect reconstruction.

6.3.3 Speech Features

Speech features such as the spectrogram [168], Mel-Frequency Cepstrum Coefficients (MFCC) [169], and Constant-Q Transform (CQT) [170] are time-frequency representation of the speech signal. Speech features are usually more concise, containing less number of samples compared to the original speech signal. The time-frequency information in speech features makes it easier to analyze content-independent low-level speech characteristics such as timbre and prosody. Therefore, speech features are widely used in a variety of speech analysis tasks. In this section, we briefly describe commonly used speech features. The input speech signal is denoted by $S[i]$, where $i \in \{0, \dots, \eta - 1\}$.

6.3.3.1 The Spectrogram

The spectrogram is the magnitude of Short Time Fourier Transform (STFT) of the original speech signal. In STFT, the speech signal is windowed into a series of same-length (denoted by

ℓ) signal segments with overlap, which results in signals $\mathcal{S}_1, \dots, \mathcal{S}_\rho$. Here, ρ denotes the number of segments. Then, Discrete Fourier Transform (DFT) is applied to each \mathcal{S}_i to generate Fourier transformed signals \tilde{S}_i for $i = 1, \dots, \rho$. The spectrogram of the speech signal can be represented as a matrix where each column is the magnitude of \tilde{S}_i . That is,

$$\text{Spectrogram}(\mathcal{S}) = \begin{bmatrix} |\tilde{S}_1(\ell/2)| & |\tilde{S}_2(\ell/2)| & \cdots & |\tilde{S}_\rho(\ell/2)| \\ |\tilde{S}_1(\ell/2 - 1)| & |\tilde{S}_2(\ell/2 - 1)| & \cdots & |\tilde{S}_\rho(\ell/2 - 1)| \\ \vdots & \vdots & \ddots & \vdots \\ |\tilde{S}_1(0)| & |\tilde{S}_2(0)| & \cdots & |\tilde{S}_\rho(0)| \end{bmatrix}. \quad (6.2)$$

More details about spectrograms are described in Section 6.4.

6.3.3.2 Mel-Frequency Cepstrum Coefficients (MFCC)

MFCC [169] is a speech feature based on STFT. The computation of MFCC also makes use of the STFT processed speech signal \tilde{S}_i . For the i -th temporal window, the power spectrum signal \mathcal{P}_i is computed by

$$\mathcal{P}_i(j) = \frac{1}{\ell/2 + 1} |\tilde{S}_i(j)|^2. \quad (6.3)$$

Then, the power spectrum signal is filtered using a Mel filterbank \mathcal{M}_i and converted to log scale, that is:

$$\text{LnMel}[P_i](j) = \ln \left(\sum_{k=0}^{\ell/2+1} \mathcal{P}_i(k) \mathcal{M}_j(k) \right). \quad (6.4)$$

An illustration of the Mel filterbank is shown in Figure 6.2, where each column corresponds to a Mel filter \mathcal{M}_i . The Mel filterbank adjusts the distribution and weights of the DFT coefficients according to the characteristics of the human hearing system. The signal $\text{LnMel}[P_i]$ is transformed using Discrete Cosine Transform (DCT) to generate MFCC:

$$\text{MFCC}[P_i](j) = \text{DCT}[\text{LnMel}[P_i]](j+1), \quad j = 0, \dots, \frac{\ell}{2}. \quad (6.5)$$

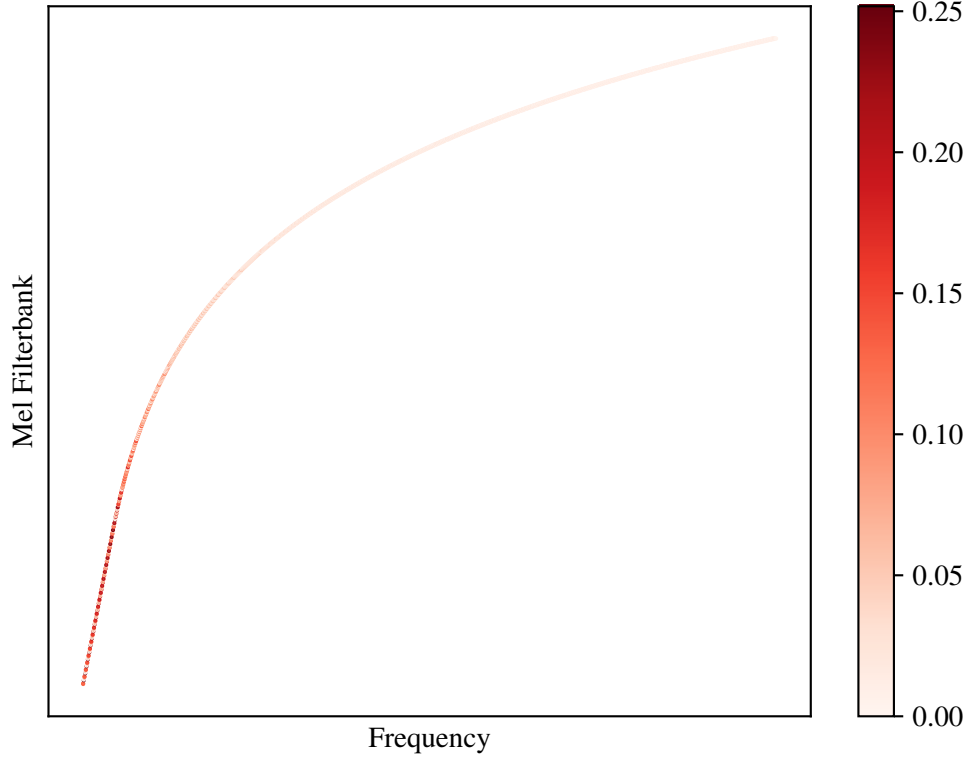


Figure 6.2. Illustration of the Mel filterbank for DFT coefficients. Note that both the frequency bins and filter magnitude are changing based on the characteristics of the human hearing system.

Note that the DC coefficient of DCT result is discarded due to its instability.

6.3.3.3 Constant-Q Transform (CQT)

Constant-Q Transform (CQT) [170] is a speech feature developed based on the characteristics of musical pitches. It is widely used in musical analysis. The center frequency of musical pitches can be modeled using exponential law [189]:

$$f_{\text{center}} = 2^{\frac{p}{12}} \cdot 440 \text{ Hz}, \quad (6.6)$$

where p is the pitch index. The low end of the range of a violin (G_3) corresponds to $p = -14$, which has a center frequency of approximately 196 Hz. The high end of the range of a violin (C_8) corresponds to $p = 39$, which has a center frequency of approximately 4186 Hz. Consider STFT

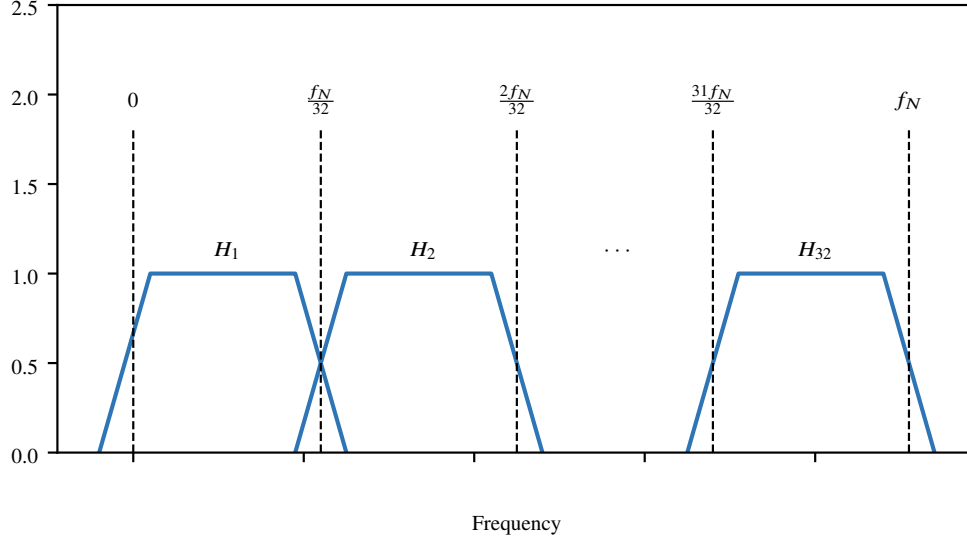


Figure 6.3. Magnitude response of the band-pass analysis filterbank used in MP3. f_N denotes the Nyquist frequency of the signal.

with a window length of 1024 samples. In STFT, the resolution of each frequency bin is the same. When it is applied a speech signal with a sampling rate of 44.1 kHz, the resolution of each frequency bin is approximately 43 Hz. The resolution is $\sim 22\%$ of the lowest violin pitch, and it is $\sim 0.8\%$ of the highest violin pitch. This shows that the resolution of STFT is usually too small for low pitches but too large for high pitches. In CQT, a series of STFTs with variable window lengths are introduced to mitigate this issue. For a given center frequency f , the window length L_f is given by

$$L_f = \frac{SQ}{f}, \quad (6.7)$$

where S is the sampling rate and Q is the quality factor. The quality factor Q is constant for different center frequencies, which is implied by the name CQT. In [170], it is suggested that $Q = 34$.

6.4 Efficient Spectrograms (E-Specs)

Spectrograms and many other speech features (e.g., MFCC and CQT) are usually computed using STFT. That is, the time-domain audio signal is partitioned into short windows of l samples with an overlap of (typically) $l/2$ samples. The spectrogram is the magnitude of the STFT coefficients

[168]. As discussed in Section 6.3.1, the MP3 compression filters the input signal into 32 bands. If STFT is used for one of the bands, the result is a patch of the STFT of the decoded audio signal from the frequency axis. Therefore, we can compute the STFT of the decoded audio signal by combining the STFT result of each band, which is more efficient. In the remainder of this section, we describe this idea in greater detail.

6.4.1 Theoretical Formulation

We denote the decoded time-domain audio signal by D . To compute STFT of D directly from D , the decoded audio signal is divided into a series of N windows D^1, \dots, D^N , where the size of each window is l . After the MP3 analysis filterbank, D is filtered into 32 bands B_1, \dots, B_{32} . The 32 bands are decimated by a factor of 32 to form bands $\hat{B}_1, \dots, \hat{B}_{32}$. Because of the decimation, the highest frequency in $\hat{B}_1, \dots, \hat{B}_{32}$ is $1/32$ of that in D . Therefore, when computing the STFT for bands $\hat{B}_1, \dots, \hat{B}_{32}$, we can use a window size l' that is much shorter than l yet still maintain a similar degree of frequency resolution. For STFT computation, the i -th decimated band \hat{B}_i will be divided into M windows $\hat{B}_i^1, \dots, \hat{B}_i^M$, where the size of each window is l' . For simplicity, we assume both l and l' are even integers.

To illustrate the efficient computation of the STFT of D , we define the truncated DFT operation of a signal S of length η (denoted by $\underline{\text{DFT}}_\eta[S]$) as follows

$$\underline{\text{DFT}}_\eta[S](i) = \begin{cases} \text{DFT}[S](i), & i = 0, 1, \dots, \frac{\eta}{2} \\ 0, & \text{otherwise,} \end{cases} \quad (6.8)$$

where the $\cdot(k)$ operator right after a sequence denotes the k -th element of the sequence. When η is clear from context, we drop it from our notations and write $\underline{\text{DFT}}[S]$ for brevity. $\underline{\text{DFT}}_\eta[S]$ is introduced because we are only interested in one side of $\text{DFT}[S]$, since S is a real signal and $\text{DFT}[S]$ is symmetric. Suppose we are computing $\underline{\text{DFT}}[D^j]$, where $j \in \{1, \dots, N\}$. For the window D^j , there must exist a neighborhood of window indices \mathcal{N}_j such that for all $j' \in \mathcal{N}_j$, the band $\hat{B}_i^{j'}$ contains samples derived from D^j (for $i = 1, \dots, 32$). Since the energy of speech signal is mostly concentrated below 3500Hz [190], we can compute DFT for the lower b ($1 \leq b \leq 32$) bands and

ignore the rest. If the band-pass filters H_1, \dots, H_{32} do not overlap, then $\underline{\text{DFT}}[D^j]$ and $\underline{\text{DFT}}[\hat{B}_i^{j'}]$ are related by

$$\underline{\text{DFT}}[D^j](k) = \sum_{i=1}^b \sum_{j' \in \mathcal{N}_j} \underline{\text{DFT}}[\hat{B}_i^{j'}] \left(k - \frac{(i-1)l'}{2} \right). \quad (6.9)$$

However, since the filters in the analysis filterbank overlap, a band can contain frequency components from adjacent bands. To cope with this problem, we interpolate and average around the boundaries of the DFT of each band. An interpolation window W_w of length $l'/2 + 1$ is constructed by sampling from the interpolation window function shown in Figure 6.4. The factor $w \in [1, 1.5]$ is used to control the “width” of the interpolation. The bigger w is, the more values around the boundaries are interpolated and averaged across windows. With W_w , we can approximate $\underline{\text{DFT}}[D^j](k)$ as follows:

$$\begin{aligned} \underline{\text{DFT}}[D^j](k) \approx \sum_{i=1}^b \sum_{j' \in \mathcal{N}_j} \left\{ \underline{\text{DFT}}[\hat{B}_i^{j'}] \left(k - (i-1) \left\lfloor \frac{l'}{2w} \right\rfloor \right) \right. \\ \left. \cdot W_w \left(k - (i-1) \left\lfloor \frac{l'}{2w} \right\rfloor \right) \right\}, \end{aligned} \quad (6.10)$$

where $\lfloor \cdot \rfloor$ denotes the floor operation. That is, $\underline{\text{DFT}}[\hat{B}_i^{j'}]$ is first weighted using the interpolation window W_w , shifted by $(i-1) \lfloor l'/2w \rfloor$, and then added to $\underline{\text{DFT}}[D^j](k)$ with an overlap of $\lceil l'(w-1)/2w \rceil$ with respect to the previous window. Here, $\lceil \cdot \rceil$ denotes the ceiling operation. Note that because of the introduction of the interpolation window W_w , the result of Equation (6.10) is an approximation of the STFT of the decoded audio signal.

The result computed using Equation (6.10) bypasses the artifacts created by the synthesis filterbank. This can be beneficial for many speech analysis tasks. Because the analysis filterbank is used in many audio compression methods [185], this approach can be extended to other audio compression techniques.

6.4.2 Efficiency Analysis

Computing the STFT using Equation (6.10) is efficient in the following two ways:

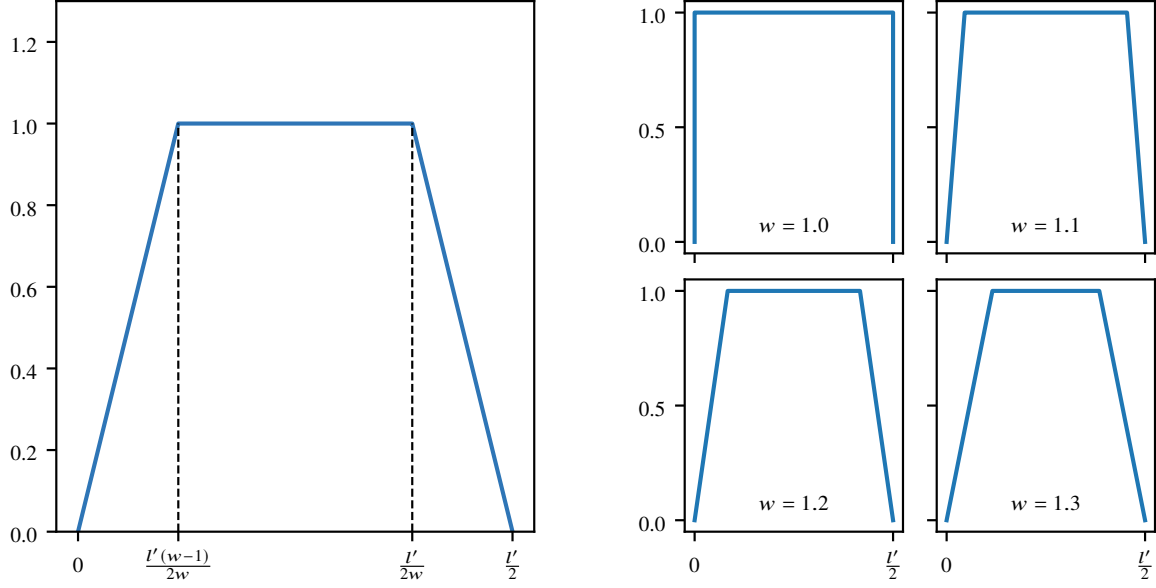


Figure 6.4. The interpolation window W_w .

Lower computational complexity. Suppose the length of the decoded time-domain signal is L . When computing STFT from D^1, \dots, D^N , the number of windows N is given by $N \approx \frac{L}{l/2}$. Since the complexity of Fast Fourier Transform (FFT) is approximately $n \log n$, where n is the number of samples, the complexity of computing STFT using this approach is approximately

$$\frac{L}{l/2} \cdot l \cdot \log l. \quad (6.11)$$

Meanwhile, using Equation (6.10) for STFT computation, the complexity is approximately

$$b \cdot \frac{L/32}{l'/2} \cdot |\mathcal{N}| \cdot l' \cdot \log l', \quad (6.12)$$

where $|\mathcal{N}|$ is the average size of the neighborhood \mathcal{N}_j . An estimate for $|\mathcal{N}|$ is given by $|\mathcal{N}| \approx \lceil l/32l' \rceil$.

The ratio between Equation (6.12) and Equation (6.11) is

$$\frac{b \cdot |\mathcal{N}|}{32} \cdot \frac{\log l'}{\log l}. \quad (6.13)$$

In our experiments (Section 6.5), we chose $l = 256$, and we observed that using $b = 16$, $l' = 12$ produced good synthetic speech detection result. In this case, Equation (6.13) is equal to

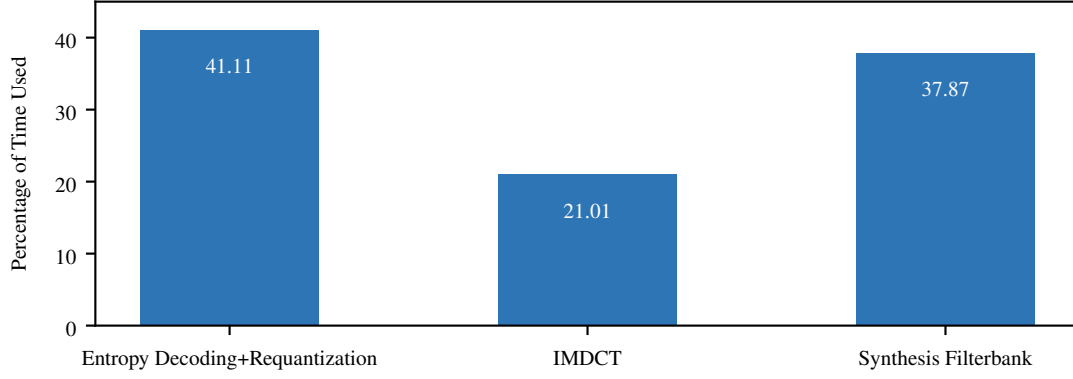


Figure 6.5. The percentage of time used by each step in the MP3 decoding process. The results were averaged after 100 repeated experiments using the `minimp3` [191] decoder.

approximately 0.224. That is, the complexity of using Equation (6.10) for STFT computation is approximately 22.40% of that using D^1, \dots, D^N .

Faster MP3 decoding. Using Equation (6.10) for STFT computation does not require the synthesis filterbank in the MP3 decoding process. We tested how long each step in the MP3 decoding process takes, and the results are shown in Figure 6.5. It can be seen that the synthesis filterbank takes approximately 37.87% of the MP3 decoding time. When using Equation (6.10) for STFT computation, this time can be saved since the decoded audio is not used.

6.4.3 Implementation of E-Spec

In practice, we avoid determining \mathcal{N}_j for each D^j . Instead, we partition each band \hat{B}_i into windows of length l' with an overlap of $l'/2$. Next, we compute the FFT of each window. The STFT of each window is weighted by W_w , shifted and added to the STFT result of the entire speech signal as per Equation (6.10). The E-Spec is the magnitude of the STFT result.

6.5 Experimental Results

We tested the performance of E-Spec in the synthetic speech detection task. In synthetic speech detection, the detector is asked to classify whether a speech signal is generated by text-to-speech systems (i.e., synthetic) or recorded from human speakers (i.e., pristine). The speech signals used in

Table 6.1. The number of parameters of each neural network architecture used in our experiments.

Model	# Parameters	Model	# Parameters
ResNet50 [195]	25.55M	EfficientNetV2 [196]	21.46M
ResNeXt50 [197]	25.02M	MobileNetV3 [198]	5.48M

our experiments are from the ASVspoof19 dataset [173]. We used the training set and the testing set of the dataset. The training set contains 22,064 speech signals; the testing set contains 54,606 speech signals. We compressed the entire dataset using MP3 with a data rate of 16kbps. We used the MP3 compressed version of the dataset to compute E-Spec. From the speech signals decoded from MP3, we extracted the three speech features: MFCC [169], CQT [170], and spectrogram [168]. The MFCC and spectrogram features are extracted using the `TorchAudio` package [192]. For MFCC computation, we set `n_mfcc` (number of MFCC coefficients) to 128 as in [193]. For spectrogram computation, we set `n_fft` (STFT window size) to 256 as in [193]. The CQT feature is extracted using the `nnAudio` package [194] with default settings.

For synthetic speech detection, we selected four popular neural network architectures to classify the extracted speech features: ResNet50 [195], EfficientNetV2 [196], ResNeXt50 [197], and MobileNetV3 [198]. The number of parameters of each model is shown in Table 6.1. Each model has been pretrained using the ImageNet1K dataset [199]. Then, they were fine-tuned on the speech features. During fine-tuning, the models were trained on the training set using the Adam optimizer [81] with a learning rate of 10^{-5} . The training was terminated after 2 epochs. The performance metrics for the models were computed on the testing set after training.

We evaluated the performance of synthetic speech detection using two metrics. The first metric is Area Under the Receiver Operating Characteristic (AUC) score, which is defined as the area under the Receiver Operating Characteristic (ROC) curve [200]. Higher AUC score indicates better performance, and a perfect detector has an AUC score of 1. The second metric is Equal Error Rate (EER) [201]. For synthetic speech detection systems, the false accept rate and the false reject rate change as the threshold changes. For a given threshold, the false accept rate will be equal to the false reject rate. The value of the false accept rate at this threshold is defined as the value of EER. Lower EER indicates better performance, and a perfect detector has an EER of 0.

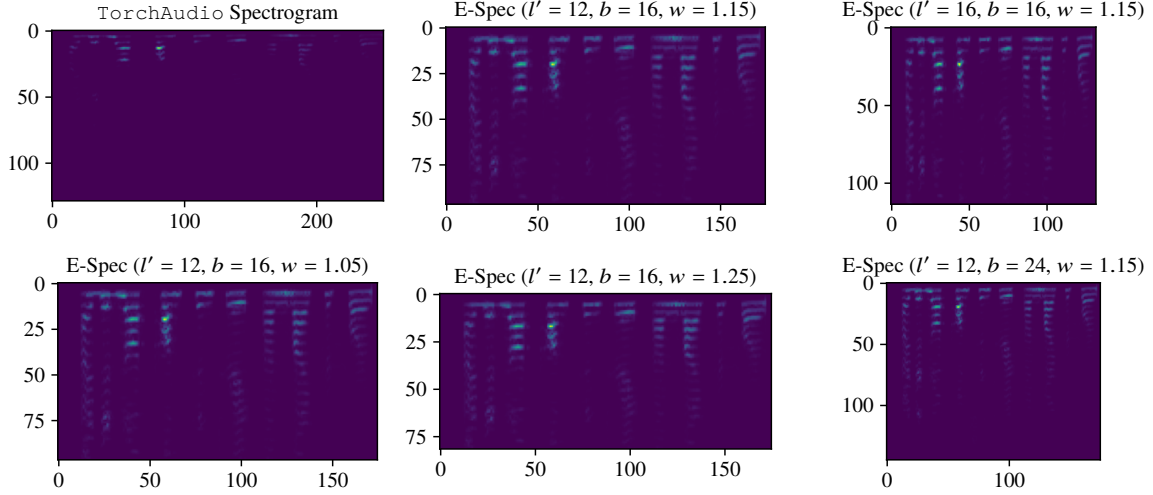


Figure 6.6. Spectrogram and E-Specs extracted from an example speech signal.

Table 6.2. The AUC scores (%) of ResNet50 [195] synthetic speech detector given different E-Spec extraction settings.

l'	b	$w = 1.05$	$w = 1.10$	$w = 1.15$	$w = 1.20$
12	12	95.65	96.83	96.23	96.32
12	16	96.93	95.94	96.89	96.57
12	20	96.47	96.69	96.41	96.94
16	12	96.55	95.83	95.95	95.37
16	16	96.72	96.01	95.66	94.66
16	20	95.84	94.77	93.56	95.19
24	12	94.67	94.54	94.46	95.04
24	16	94.31	94.71	93.97	93.77
24	20	95.19	93.54	94.23	94.53

E-Spec extraction settings. We investigated the influence of the choice of l' , b , and w over the performance of synthetic speech detection. We changed the values of l' , b , and w in the computation of E-Spec, and used the E-Spec to train synthetic speech detectors based on ResNet50 [195]. The spectrograms and E-Specs extracted from an example speech signal are shown in Figure 6.6. The AUC score and EER of different E-Spec extraction settings are shown in Tables 6.2 and 6.3, respectively. It can be seen that $l' = 12$, $b = 16$, and $w = 1.15$ led to the best EER on the testing set.

Performance of E-Spec across neural network architectures. To show that E-Spec can achieve good synthetic speech detection performance across different model architecture, we fixed the E-Spec extraction settings to $l' = 12$, $b = 16$, $w = 1.15$. Then, we acquired the synthetic speech

Table 6.3. The EER (%) of ResNet50 [195] synthetic speech detector given different E-Spec extraction settings.

l'	b	$w = 1.05$	$w = 1.10$	$w = 1.15$	$w = 1.20$
12	12	11.66	9.92	10.49	10.27
12	16	9.54	10.98	9.13	10.00
12	20	10.42	10.15	10.56	9.90
16	12	10.21	10.55	11.34	11.73
16	16	9.51	11.48	11.85	12.96
16	20	11.32	13.11	14.23	12.31
24	12	12.75	12.48	12.51	11.88
24	16	13.23	12.24	13.59	13.60
24	20	11.86	14.01	12.84	12.71

Table 6.4. The synthetic speech detection AUC score (%) of speech features computed from decoded speech and E-Spec across different neural network architectures.

Network Architecture	Speech Features			
	MFCC	CQT	Spectrogram	E-Spec
ResNet50 [195]	94.14	94.45	95.98	96.89
EfficientNetV2 [196]	91.52	93.95	96.13	96.20
ResNeXt50 [197]	93.87	94.22	96.56	97.26
MobileNetV3 [198]	92.79	94.25	94.29	94.49
Average	93.08	94.22	95.74	96.21

detection performance using speech features computed from decoded speech signals, and compared them against the performance of E-Spec across different architectures. The AUC score and EER of the models are shown in Tables 6.3 and 6.4, respectively. It can be seen that E-Spec achieved the best EER on all models except for MobileNetV3. E-Spec had the best overall synthetic speech detection EER.

6.6 Conclusion

In this work, we proposed an efficient way to compute spectrogram for MP3 compressed audio, which is denoted by E-Spec. E-Spec can reduce the computation complexity of spectrogram by ~ 77.60 p.p. for MP3 compressed audio signals. E-Spec also does not require the MP3 synthesis filterbank, which saves ~ 37.87 p.p. of MP3 decoding time. We tested the performance of E-Spec in

Table 6.5. The synthetic speech detection EER (%) of speech features computed from decoded speech and E-Spec across different neural network architectures.

Network Architecture	Speech Features			
	MFCC	CQT	Spectrogram	E-Spec
ResNet50 [195]	13.57	13.12	10.29	9.13
EfficientNetV2 [196]	15.71	13.47	11.00	10.35
ResNeXt50 [197]	14.17	13.54	9.66	9.41
MobileNetV3 [198]	14.84	12.83	12.16	12.62
Average	14.57	13.24	10.78	10.38

the synthetic speech detection task. The experimental results showed that E-Spec achieved the best overall synthetic speech performance compared to speech features from the decoded audio signal across different neural network architectures. E-Spec does not contain the reconstruction artifacts created by the MP3 synthesis filterbank, which can be useful for many audio forensics tasks. E-Spec can be formulated as an approximation to STFT, which can be potentially used to accelerate the speed of a variety of speech analysis tasks.

In the future, we will evaluate the performance E-Spec in other audio analysis tasks. We will try to compute other audio features such as MFCC and CQT based on E-Spec, and assess their performance in different tasks. We will try to extend our proposed approach to other audio compression techniques such as AAC.

7. SUMMARY AND FUTURE WORK

7.1 Forensic Analysis of Video Files Using Metadata

In this chapter, we proposed a video forensics approach for MP4 video files based on metadata embedded in video files. Our improved metadata extraction and feature representation scheme allows one to represent more metadata information in a compact feature vector. We use feature selection, dimensionality reduction, and nearest neighbor classification techniques to form interpretable and reliable decision rules for multiple video forensics scenarios. Our approach achieves better performance than other methods.

The performance of our method in many of the scenarios indicates that we need to increase our video forensics dataset to include more difficult cases. Our research also exposed the limitation of metadata-based forensics methods, namely its failure to analyze videos from specific social networks such as TikTok and WeiBo. This is a significant disadvantage compared to pixel-based methods. In the future, we plan to continue exploring the potential of metadata-based video forensics by adding the ability to parse more manufacturer-specific data models (e.g., Canon’s CNTH tags [41]) and by looking into lower-level metadata in the distribution of audio/video samples as well as location of key frames in the video stream. We hope that metadata-based video forensics methods can be proved to be reliable in more forensic scenarios.

7.2 Forensic Analysis and Localization of Multiply Compressed MP3 Audio Using Transformers

We proposed a multiple MP3 compression temporal localization method based on transformer neural networks that uses MP3 compressed data. Our proposed method localizes multiple compression at the frame level. The experiment results showed that our method had the best performance compared to other approaches and was robust against many MP3 compression settings. In the future, we will examine extending this approach to other compression methods such as AAC. We will also investigate the use of stereo channels as well as the second granule in MP3 compressed frames. We will generalize the concept of multiple compression detection to compression history detection. That is, to find the number of compressions and the types of compression used. Knowing the compression history can greatly enhance the interpretability for audio forensics.

7.3 H4VDM: H.264 Video Device Matching

In this paper we proposed an H.264-based open-set Video Device Matching (VDM) method known as H4VDM. H4VDM uses transformer neural networks to process five types of data from the H.264 decoded frames and the Group of Picture (GOP)s. We trained and tested the H4VDM-B model on datasets generated from the VISION dataset [132]. The experimental results showed that H4VDM demonstrated good VDM performance on unseen devices.

Despite the good performance, H4VDM has still room for improvement. When selecting hyperparameters, we greedily used the model that had the best performance on Dataset D1. The selected model may not have the best overall performance across all datasets. Some important H.264 codec information such as motion vectors and true prediction residuals were not used in H4VDM. The small number of devices in the VISION dataset also limited the performance evaluation of H4VDM. On datasets D1–D4, less dataset bias is introduced in training/testing split, but the performance of H4VDM is relatively low due to small number of devices in the training set. On datasets D5–D7, the performance of H4VDM is higher, but the influence of dataset bias is stronger due to the small number of devices in the testing set, which resulted in fluctuating testing performance. A dataset with more devices is required to evaluate the performance of H4VDM more comprehensively.

In future work, we will examine the use of motion vectors and prediction residuals. We will develop methods that use popular deep learning frameworks efficiently so that the training speed is less constrained by hardware I/O speed. We will collect video data from more video capturing devices for future video forensics research. We are investigating other video compression techniques including H.265, H.266, VP9, and AV1.

7.4 MTN: Forensic Analysis of MP4 Video Files Using Graph Neural Networks

In this chapter, we propose MTN for video forensics analysis based on end-to-end Graph Neural Networks (GNNs). MTN uses the MP4 tree information to make decisions, it does not require any pixel data. We devise an Self-Supervised Learning (SSL) scheme to pretrain MTN, which allows it to generate semantic-preserving node embeddings. We also propose a data augmentation technique for MP4 trees to help train MTN in data-scarce scenarios. The experimental results showed that MTN achieved good performance in video forensics analysis tasks. It is shown that MTN can gain

more comprehensive understanding about the MP4 trees and is more robust to the evidence removal attack compared to existing methods. We also demonstrated that the SSL scheme and the data augmentation technique can reduce training iterations, reduce performance fluctuation, and improve the performance.

In the future, we will examine the performance of MTN in more video forensics analysis tasks. We will evaluate the scalability of MTN by analyzing large real-world datasets.

7.5 Extracting Efficient Spectrograms From MP3 Compressed Speech Signals for Synthetic Speech Detection

In this work, we proposed an efficient way to compute spectrogram for MP3 compressed audio, which is denoted by Efficient Spectrogram (E-Spec). E-Spec can reduce the computation complexity of spectrogram by ~ 77.60 percentage points (p.p.) for MP3 compressed audio signals. E-Spec also does not require the MP3 synthesis filterbank, which saves ~ 37.87 p.p. of MP3 decoding time. We tested the performance of E-Spec in the synthetic speech detection task. The experimental results showed that E-Spec achieved the best overall synthetic speech performance compared to speech features from the decoded audio signal across different neural network architectures. E-Spec does not contain the reconstruction artifacts created by the MP3 synthesis filterbank, which can be useful for many audio forensics tasks. E-Spec can be formulated as an approximation to Short Time Fourier Transform (STFT), which can be potentially used to accelerate the speed of a variety of speech analysis tasks.

In the future, we will evaluate the performance E-Spec in other audio analysis tasks. We will try to compute other audio features such as Mel-Frequency Cepstrum Coefficients (MFCC) and Constant-Q Transform (CQT) based on E-Spec, and assess their performance in different tasks. We will try to extend our proposed approach to other audio compression techniques such as AAC.

7.6 Contributions Of This Work

In this work, we developed new methods for multimedia forensics using metadata. The main contributions of the work are listed as follows:

- We proposed a video forensics method using the metadata in MP4/MOV video containers, which allows the visualization of video metadata features and achieves good performance in a wide variety of video forensics tasks
- We proposed an MP3 multiple compression localization method using transformer neural networks, which outperforms other methods in terms of both localization granularity and accuracy
- We proposed a video device matching method for H.264 video sequences, which can achieve open-set video device matching with high accuracy
- We proposed a Graph Neural Network (GNN) based method for the analysis of MP4/MOV metadata trees; the GNN is trained using Self-Supervised Learning (SSL), which increases the robustness of the analysis and makes it capable of handling missing/unseen data
- We proposed a technique to compute the spectrogram feature from MP3 compressed signals without fully decoding them; the proposed technique decreases the complexity of speech feature computation by $\sim 77.6\%$ and saves $\sim 37.87\%$ of MP3 decoding time; the resulting spectrogram features lead to higher synthetic speech detection performance

REFERENCES

- [1] C. Jee, *An Indian politician is using deepfake technology to win new voters*. [Online]. Available: <https://www.technologyreview.com/2020/02/19/868173/an-indian-politician-is-using-deepfakes-to-try-and-win-voters/>.
- [2] R. T. Garcia, *Deepfakes Are Being Used to Puncture Politicians Bluster*. [Online]. Available: <https://ffwd.medium.com/deepfakes-are-being-used-to-puncture-politicians-bluster-e4bb4473841>.
- [3] S. Milani, M. Fontani, P. Bestagini, M. Barni, A. Piva, M. Tagliasacchi, and S. Tubaro, “An overview on video forensics,” *APSIPA Transactions on Signal and Information Processing*, vol. 1, e2, 2012. DOI: [10.1017/ATSIP.2012.2](https://doi.org/10.1017/ATSIP.2012.2).
- [4] S. Bayram, H. T. Sencar, and N. Memon, “Video Copy Detection Based on Source Device Characteristics: A Complementary Approach to Content-Based Methods,” *Proceedings of the ACM International Conference on Multimedia Information Retrieval*, pp. 435–442, Oct. 2008, Vancouver, British Columbia, Canada. DOI: [10.1145/1460096.1460167](https://doi.org/10.1145/1460096.1460167).
- [5] M. Koopman, A. Macarulla Rodriguez, and Z. Geradts, “Detection of Deepfake Video Manipulation,” *Proceedings of the Irish Machine Vision and Image Processing Conference*, pp. 133–136, Aug. 2018, Belfast, United Kingdom.
- [6] H. H. Nguyen, F. Fang, J. Yamagishi, and I. Echizen, “Multi-task learning for detecting and segmenting manipulated facial images and videos,” *Proceedings of the IEEE International Conference on Biometrics*, Sep. 2019, Tampa, Florida. DOI: [10.1109/BTAS46853.2019.9185974](https://doi.org/10.1109/BTAS46853.2019.9185974).
- [7] E. Sabir, J. Cheng, A. Jaiswal, W. AbdAlmageed, I. Masi, and P. Natarajan, “Recurrent Convolutional Strategies for Face Manipulation Detection in Videos,” *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, Jun. 2019, Long Beach, California, USA. DOI: [10.1109/IJCB48548.2020.9304936](https://doi.org/10.1109/IJCB48548.2020.9304936).
- [8] M. Brundage, S. Avin, J. Clark, H. Toner, P. Eckersley, B. Garfinkel, A. Dafoe, P. Scharre, T. Zeitzoff, B. Filar, H. Anderson, H. Roff, G. C. Allen, J. Steinhardt, C. Flynn, S. O hEigearthaigh, S. Beard, H. Belfield, S. Farquhar, C. Lyle, R. Crotoft, O. Evans, M. Page, J. Bryson, R. Yampolskiy, and D. Amode, “The Malicious Use of Artificial Intelligence : Forecasting, Prevention, and Mitigation,” *arXiv preprint arXiv:1802.07228*, Feb. 2018. DOI: [10.48550/arXiv.1802.07228](https://doi.org/10.48550/arXiv.1802.07228). [Online]. Available: <https://arxiv.org/abs/1802.07228v1>.
- [9] Committee On Ethics, *Intentional Use of Audio-Visual Distortions and Deep Fakes*. [Online]. Available: <https://ethics.house.gov/campaign-activity-pink-sheets/intentional-use-audio-visual-distortions-deep-fakes>.

- [10] S. Grobman, *McAfee Labs 2020 Threats Predictions Report*. [Online]. Available: <https://www.mcafee.com/blogs/other-blogs/mcafee-labs/mcafee-labs-2020-threats-predictions-report/>.
- [11] D. Ruiz, *Deepfakes laws and proposals flood US*. [Online]. Available: <https://blog.malwarebytes.com/artificial-intelligence/2020/01/deepfakes-laws-and-proposals-flood-us/>.
- [12] J. Umawing, *The face of tomorrows cybercrime: Deepfake ransomware explained*. [Online]. Available: <https://www.terabitweb.com/2020/06/26/the-face-of-tomorrows-cybercrime-deepfake-ransomware-explained/>.
- [13] D. Webb, *Avatarify: Create Real Time Deepfakes for Video Calls*. [Online]. Available: <https://ccm.net/faq/64681-avatarify-video-call-deepfakes>.
- [14] D. Güera, S. Baireddy, P. Bestagini, S. Tubaro, and E. J. Delp, “We Need No Pixels: Video Manipulation Detection Using Stream Descriptors,” *arXiv preprint arXiv:1906.08743*, Jun. 2019. doi: 10.48550/arXiv.1906.08743. [Online]. Available: <https://arxiv.org/abs/1906.08743>.
- [15] M. Iuliani, D. Shullani, M. Fontani, S. Meucci, and A. Piva, “A Video Forensic Framework for the Unsupervised Analysis of MP4-Like File Container,” *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 3, pp. 635–645, Mar. 2019. doi: 10.1109/TIFS.2018.2859760.
- [16] K. Jack, *Chapter 13 - MPEG-2*. Burlington, MA: Newnes, 2007, pp. 577–737. doi: 10.1016/B978-075068395-1/50013-4. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/B9780750683951500134>.
- [17] P. Yang, D. Baracchi, M. Iuliani, D. Shullani, R. Ni, Y. Zhao, and A. Piva, “Efficient Video Integrity Analysis Through Container Characterization,” *IEEE Journal of Selected Topics in Signal Processing*, vol. 14, no. 5, pp. 947–954, Aug. 2020. doi: 10.1109/JSTSP.2020.3008088.
- [18] Google LLC, *Android Developers: Supported media formats*. [Online]. Available: <https://developer.android.com/guide/topics/media/media-formats>.
- [19] Sony Corporation, *FDR-AX40 Specifications*. [Online]. Available: <https://www.sony.co.in/electronics/handycam-camcorders/fdr-ax40/specifications>.
- [20] A. York, *Always Up-to-Date Guide to Social Media Video Specs*. [Online]. Available: <https://sproutsocial.com/insights/social-media-video-specs-guide/>.
- [21] ISO, *ISO/IEC 14496-12:2020 - Information technology - Coding of audio-visual objects - Part 12: ISO base media file format*. [Online]. Available: <https://www.iso.org/standard/74428.html>.
- [22] Apple Inc., *Classic Version of the QuickTime File Format Specification*. [Online]. Available: <https://developer.apple.com/standards/classic-quicktime/>.

- [23] R. Schafer and T. Sikora, “Digital video coding standards and their role in video communications,” *Proceedings of the IEEE*, vol. 83, no. 6, pp. 907–924, Jun. 1995. doi: [10.1109/5.387092](https://doi.org/10.1109/5.387092).
- [24] D. Vázquez-Padín, M. Fontani, D. Shullani, F. Pérez-González, A. Piva, and M. Barni, “Video Integrity Verification and GOP Size Estimation Via Generalized Variation of Prediction Footprint,” *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1815–1830, 2020. doi: [10.1109/TIFS.2019.2951313](https://doi.org/10.1109/TIFS.2019.2951313).
- [25] H. Yao, R. Ni, and Y. Zhao, “Double compression detection for H. 264 videos with adaptive GOP structure,” *Multimedia Tools and Applications*, vol. 79, no. 9, pp. 5789–5806, Mar. 2020. doi: [10.1007/s11042-019-08306-5](https://doi.org/10.1007/s11042-019-08306-5).
- [26] ISO, *ISO/IEC 13818-1:2019 - Information technology - Generic coding of moving pictures and associated audio information - Part 1: Systems*. [Online]. Available: <https://www.iso.org/standard/75928.html>.
- [27] G. J. Sullivan, J. Ohm, W. Han, and T. Wiegand, “Overview of the High Efficiency Video Coding (HEVC) standard,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, Dec. 2012. doi: [10.1109/TCSVT.2012.2221191](https://doi.org/10.1109/TCSVT.2012.2221191).
- [28] G. J. Sullivan and T. Wiegand, “Video Compression - From Concepts to the H.264/AVC standard,” *Proceedings of the IEEE*, vol. 93, no. 1, pp. 18–31, Jan. 2005. doi: [10.1109/JPROC.2004.839617](https://doi.org/10.1109/JPROC.2004.839617).
- [29] ISO, *ISO/IEC 14496-14:2020 - Information technology - Coding of audio-visual objects - Part 14: MP4 file format*. [Online]. Available: <https://www.iso.org/standard/79110.html>.
- [30] Apple Inc., *QuickTime File Format Specification–Metadata*. [Online]. Available: <https://developer.apple.com/library/archive/documentation/QuickTime/QTFF/Metadata/Metadata.html>.
- [31] Apple Inc., *QuickTime File Format Specification–Movie Atoms*. [Online]. Available: <https://developer.apple.com/library/archive/documentation/QuickTime/QTFF/QTFFChap2/qtff2.html>.
- [32] Leo van Stee, *On date, time, location and other metadata in MP4/MOV files*. [Online]. Available: <https://leo-van-stee.github.io/>.
- [33] J. Shi and J. Malik, “Normalized cuts and image segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, Aug. 2000. doi: [10.1109/34.868688](https://doi.org/10.1109/34.868688).
- [34] J. Friedman, T. Hastie, R. Tibshirani, *et al.*, *The elements of statistical learning*. New York, USA: Springer, 2009, vol. 1, pp. 106–119.

- [35] D. Shullani, M. Fontani, M. Iuliani, O. Alshaya, and A. Piva, “VISION: a video and image dataset for source identification,” *EURASIP Journal on Information Security*, vol. 2017, p. 15, Oct. 2017. DOI: [10.1186/s13635-017-0067-2](https://doi.org/10.1186/s13635-017-0067-2).
- [36] Avidemux contributors, *Avidemux*. [Online]. Available: <http://avidemux.sourceforge.net/>.
- [37] P. Harvey, *ExifTool by Phil Harvey*. [Online]. Available: <https://exiftool.org/>.
- [38] FFmpeg contributors, *FFmpeg*. [Online]. Available: <https://www.ffmpeg.org/>.
- [39] Kdenlive contributors, *Kdenlive*. [Online]. Available: <https://kdenlive.org/en/>.
- [40] Adobe Inc., *Professional video editor & video maker - Adobe Premiere Pro*. [Online]. Available: <https://www.adobe.com/products/premiere.html>.
- [41] Exiftool contributors, *Canon Tags*. [Online]. Available: <https://exiftool.org/TagNames/Canon.html>.
- [42] M. Filippone, F. Camastra, F. Masulli, and S. Rovetta, “A Survey of Kernel and Spectral Methods for Clustering,” *Pattern recognition*, vol. 41, no. 1, pp. 176–190, 2008. DOI: [10.1016/j.patcog.2007.05.018](https://doi.org/10.1016/j.patcog.2007.05.018).
- [43] A. Ng, M. Jordan, and Y. Weiss, “On Spectral Clustering: Analysis and an Algorithm,” *Advances in neural information processing systems*, vol. 14, 2001.
- [44] F. R. Chung, *Spectral Graph theory*. American Mathematical Soc., 1997, vol. 92. DOI: [10.1090/cbms/092](https://doi.org/10.1090/cbms/092).
- [45] J. MacQueen, “Some Methods for Classification and Analysis of Multivariate Observations,” *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14, pp. 281–297, 1967, Berkeley, CA, USA.
- [46] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2009. DOI: [10.1007/978-0-387-84858-7](https://doi.org/10.1007/978-0-387-84858-7).
- [47] Q. Gu, Z. Li, and J. Han, “Linear Discriminant Dimensionality Reduction,” *Proceedings of the Machine Learning and Knowledge Discovery in Databases: European Conference*, pp. 549–564, 2011, Athens, Greece. DOI: [10.1007/978-3-642-23780-5_45](https://doi.org/10.1007/978-3-642-23780-5_45).
- [48] K. Fukunaga, *Introduction to Statistical Pattern Recognition*. Elsevier, 2013. DOI: [10.1016/C2009-0-27872-X](https://doi.org/10.1016/C2009-0-27872-X).

- [49] R. Prenger, R. Valle, and B. Catanzaro, “Waveglow: A flow-based generative network for speech synthesis,” *Proceedings of 2019 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 3617–3621, 2019, Brighton, UK. doi: [10.1109/ICASSP.2019.8683143](https://doi.org/10.1109/ICASSP.2019.8683143).
- [50] Y. Ren, C. Hu, X. Tan, T. Qin, S. Zhao, Z. Zhao, and T.-Y. Liu, “Fastspeech 2: Fast and high-quality end-to-end text to speech,” *arXiv preprint arXiv:2006.04558*, 2020. doi: [10.48550/arXiv.2006.04558](https://doi.org/10.48550/arXiv.2006.04558).
- [51] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, and I. Sutskever, “Jukebox: A generative model for music,” *arXiv preprint arXiv:2005.00341*, 2020. doi: [10.48550/arXiv.2005.00341](https://doi.org/10.48550/arXiv.2005.00341).
- [52] C. Borrelli, P. Bestagini, F. Antonacci, A. Sarti, and S. Tubaro, “Synthetic speech detection through short-term and long-term prediction traces,” *EURASIP Journal on Information Security*, vol. 2021, no. 2, pp. 1–14, 2021. doi: [10.1186/s13635-021-00116-3](https://doi.org/10.1186/s13635-021-00116-3).
- [53] R. Yang, Y.-Q. Shi, and J. Huang, “Defeating fake-quality MP3,” *Proceedings of the 11th ACM Workshop on Multimedia and Security*, pp. 117–124, 2009, Princeton, NJ, USA. doi: [10.1145/1597817.1597838](https://doi.org/10.1145/1597817.1597838).
- [54] Q. Liu, A. H. Sung, and M. Qiao, “Detection of double MP3 compression,” *Cognitive Computation*, vol. 2, no. 4, pp. 291–296, 2010. doi: [10.1007/s12559-010-9045-4](https://doi.org/10.1007/s12559-010-9045-4).
- [55] *ISO/IEC 13818-3:1995 - Information technology – Generic coding of moving pictures and associated audio information – Part 3: Audio*, <https://www.iso.org/standard/22991.html>.
- [56] H. Musmann, “Genesis of the MP3 audio coding standard,” *IEEE Transactions on Consumer Electronics*, vol. 52, no. 3, pp. 1043–1049, 2006. doi: [10.1109/TCE.2006.1706505](https://doi.org/10.1109/TCE.2006.1706505).
- [57] *ISO/IEC 13818-7:1997 - Information technology – Generic coding of moving pictures and associated audio information – Part 7: Advanced Audio Coding (AAC)*, <https://www.iso.org/standard/25040.html>.
- [58] K. Brandenburg, “MP3 and AAC explained,” *Proceedings of the AES 17th International Conference on High-Quality Audio Coding*, 1999, Signa, Italy.
- [59] R. Yang, Y. Q. Shi, and J. Huang, “Detecting double compression of audio signal,” *Media Forensics and Security II*, vol. 7541, pp. 200–209, 2010. doi: [10.1117/12.838695](https://doi.org/10.1117/12.838695).
- [60] M. Qiao, A. H. Sung, and Q. Liu, “Improved detection of MP3 double compression using content-independent features,” *Proceedings of 2013 IEEE International Conference on Signal Processing, Communication and Computing*, pp. 1–4, 2013, KunMing, China. doi: [10.1109/ICSPCC.2013.6664121](https://doi.org/10.1109/ICSPCC.2013.6664121).

- [61] P. Ma, R. Wang, D. Yan, and C. Jin, “Detecting double-compressed MP3 with the Same Bit-rate,” *Journal of Software*, vol. 9, no. 10, pp. 2522–2527, 2014.
- [62] P. Ma, R. Wang, D. Yan, and C. Jin, “A Huffman Table Index Based Approach to Detect Double MP3 Compression,” in *Digital-Forensics and Watermarking*, Berlin, Heidelberg, 2014, pp. 258–271. DOI: [10.1007/978-3-662-43886-2_19](https://doi.org/10.1007/978-3-662-43886-2_19).
- [63] D. Yan, R. Wang, J. Zhou, C. Jin, and Z. Wang, “Compression history detection for MP3 audio,” *KSII Transactions on Internet and Information Systems (TIIS)*, vol. 12, no. 2, pp. 662–675, 2018. DOI: [10.3837/tiis.2018.02.007](https://doi.org/10.3837/tiis.2018.02.007).
- [64] T. Bianchi, A. De Rosa, M. Fontani, G. Rocciolo, and A. Piva, “Detection and localization of double compression in MP3 audio tracks,” *EURASIP Journal on information Security*, vol. 2014, no. 10, 2014. DOI: [10.1186/1687-417X-2014-10](https://doi.org/10.1186/1687-417X-2014-10).
- [65] D. Luo, W. Cheng, H. Yuan, W. Luo, and Z. Liu, “Compression Detection of Audio Waveforms Based on Stacked Autoencoders,” in *Artificial Intelligence and Security*, Cham, 2020, pp. 393–404. DOI: [10.1007/978-3-030-57881-7_35](https://doi.org/10.1007/978-3-030-57881-7_35).
- [66] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in neural information processing systems*, vol. 30, 2017, pp. 5998–6008.
- [67] N. Jayant, J. Johnston, and R. Safranek, “Signal compression based on models of human perception,” *Proceedings of the IEEE*, vol. 81, no. 10, pp. 1385–1422, 1993. DOI: [10.1109/5.241504](https://doi.org/10.1109/5.241504).
- [68] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, “Language Models are Few-Shot Learners,” in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 1877–1901.
- [69] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, “An image is worth 16x16 words: Transformers for image recognition at scale,” *arXiv preprint arXiv:2010.11929*, 2020. DOI: [10.48550/arXiv.2010.11929](https://doi.org/10.48550/arXiv.2010.11929).
- [70] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, “End-to-End Object Detection with Transformers,” *Proceedings of 2020 European Conference on Computer Vision*, pp. 213–229, 2020. DOI: [10.1007/978-3-030-58452-8_13](https://doi.org/10.1007/978-3-030-58452-8_13).
- [71] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, *et al.*, “Highly accurate protein structure prediction with AlphaFold,” *Nature*, vol. 596, no. 7873, pp. 583–589, 2021. DOI: [10.1038/s41586-021-03819-2](https://doi.org/10.1038/s41586-021-03819-2).

- [72] S. Liu and W. Deng, “Very deep convolutional neural network based image classification using small training sample size,” *Proceedings of 2015 3rd IAPR Asian Conference on Pattern Recognition*, pp. 730–734, 2015. doi: [10.1109/ACPR.2015.7486599](https://doi.org/10.1109/ACPR.2015.7486599).
- [73] R. Raissi, *The theory behind MP3*, 2002. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.113.6804>.
- [74] P. Sripada, “MP3 decoder in theory and practice,” M.S. thesis, Blekinge Institute of Technology, Ronneby, Sweden, Mar. 2006. [Online]. Available: <https://www.diva-portal.org/smash/get/diva2:830195/FULLTEXT01.pdf>.
- [75] S. Ruder, “An overview of gradient descent optimization algorithms,” *arXiv preprint arXiv:1609.04747*, 2016. doi: [10.48550/arXiv.1609.04747](https://doi.org/10.48550/arXiv.1609.04747).
- [76] D. Hendrycks and K. Gimpel, “Gaussian error linear units (GELUs),” *arXiv preprint arXiv:1606.08415*, 2016. doi: [10.48550/arXiv.1606.08415](https://doi.org/10.48550/arXiv.1606.08415).
- [77] K. Ito and L. Johnson, *The LJ Speech Dataset*, 2017. [Online]. Available: <https://keithito.com/LJ-Speech-Dataset/>.
- [78] G. Tzanetakis and P. Cook, “Musical genre classification of audio signals,” *IEEE Transactions on Speech and Audio Processing*, vol. 10, no. 5, pp. 293–302, 2002. doi: [10.1109/TSA.2002.800560](https://doi.org/10.1109/TSA.2002.800560).
- [79] C. Hawthorne, A. Stasyuk, A. Roberts, I. Simon, C.-Z. A. Huang, S. Dieleman, E. Elsen, J. Engel, and D. Eck, “Enabling Factorized Piano Music Modeling and Generation with the MAESTRO Dataset,” *Proceedings of the International Conference on Learning Representations*, 2019.
- [80] *FFmpeg MP3 encoding guide*. [Online]. Available: <https://trac.ffmpeg.org/wiki/Encode/MP3>.
- [81] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014. doi: [10.48550/arXiv.1412.6980](https://doi.org/10.48550/arXiv.1412.6980).
- [82] M. Levandowsky and D. Winter, “Distance between sets,” *Nature*, vol. 234, no. 5323, pp. 34–35, 1971. doi: [10.1038/234034a0](https://doi.org/10.1038/234034a0).
- [83] D. M. Powers, “Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation,” *arXiv preprint arXiv:2010.16061*, 2020. doi: [10.48550/arXiv.2010.16061](https://doi.org/10.48550/arXiv.2010.16061).
- [84] K. H. Brodersen, C. S. Ong, K. E. Stephan, and J. M. Buhmann, “The Balanced Accuracy and Its Posterior Distribution,” *2010 20th International Conference on Pattern Recognition*, pp. 3121–3124, 2010, Istanbul, Turkey. doi: [10.1109/ICPR.2010.764](https://doi.org/10.1109/ICPR.2010.764).

- [85] K. Kurosawa, K. Kuroki, and N. Saitoh, “CCD fingerprint method-identification of a video camera from videotaped images,” *Proceedings of the International Conference on Image Processing*, vol. 3, 537–540 vol.3, 1999, Kobe, Japan. doi: [10.1109/ICIP.1999.817172](https://doi.org/10.1109/ICIP.1999.817172).
- [86] J. Lukas, J. Fridrich, and M. Goljan, “Digital camera identification from sensor pattern noise,” *IEEE Transactions on Information Forensics and Security*, vol. 1, no. 2, pp. 205–214, 2006. doi: [10.1109/TIFS.2006.873602](https://doi.org/10.1109/TIFS.2006.873602).
- [87] F. Marra, G. Poggi, C. Sansone, and L. Verdoliva, “Blind PRNU-Based Image Clustering for Source Identification,” *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 9, pp. 2197–2211, 2017. doi: [10.1109/TIFS.2017.2701335](https://doi.org/10.1109/TIFS.2017.2701335).
- [88] S. Mandelli, D. Cozzolino, P. Bestagini, L. Verdoliva, and S. Tubaro, “CNN-Based Fast Source Device Identification,” *IEEE Signal Processing Letters*, vol. 27, pp. 1285–1289, 2020. doi: [10.1109/LSP.2020.3008855](https://doi.org/10.1109/LSP.2020.3008855).
- [89] M. Iuliani, M. Fontani, D. Shullani, and A. Piva, “Hybrid reference-based video source identification,” *Sensors*, vol. 19, no. 3, p. 649, 2019. doi: [10.3390/s19030649](https://doi.org/10.3390/s19030649).
- [90] S. Mandelli, P. Bestagini, L. Verdoliva, and S. Tubaro, “Facing Device Attribution Problem for Stabilized Video Sequences,” *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 14–27, 2020. doi: [10.1109/TIFS.2019.2918644](https://doi.org/10.1109/TIFS.2019.2918644).
- [91] S. Mandelli, P. Bestagini, S. Tubaro, D. Cozzolino, and L. Verdoliva, “Blind Detection and Localization of Video Temporal Splicing Exploiting Sensor-Based Footprints,” *Proceedings of the European Signal Processing Conference*, pp. 1362–1366, 2018, Rome, Italy. doi: [10.23919/EUSIPCO.2018.8553511](https://doi.org/10.23919/EUSIPCO.2018.8553511).
- [92] International Organization for Standardization, *ISO/IEC 14496-10:2020 information technology—coding of audio-visual objects—part 10: Advanced video coding*, <https://www.iso.org/standard/75400.html>.
- [93] I. E. Richardson, *The H. 264 advanced video compression standard*. John Wiley & Sons, 2011.
- [94] Z. Xiang, J. Horváth, S. Baireddy, P. Bestagini, S. Tubaro, and E. J. Delp, “Forensic Analysis of Video Files Using Metadata,” *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 1042–1051, 2021, Nashville, TN, USA. doi: [10.1109/CVPRW53098.2021.00115](https://doi.org/10.1109/CVPRW53098.2021.00115).
- [95] R. Ramos López, E. Almaraz Luengo, A. L. Sandoval Orozco, and L. J. G. Villalba, “Digital Video Source Identification Based on Container’s Structure Analysis,” *IEEE Access*, vol. 8, pp. 36 363–36 375, 2020. doi: [10.1109/ACCESS.2020.2971785](https://doi.org/10.1109/ACCESS.2020.2971785).

- [96] E. Altinisik and H. T. Sencar, "Camera Model Identification Using Container and Encoding Characteristics of Video Files," *arXiv preprint arXiv:2201.02949*, 2022. DOI: [10.48550/arXiv.2201.02949](https://doi.org/10.48550/arXiv.2201.02949).
- [97] W. J. Scheirer, A. Rocha, A. Sapkota, and T. E. Boult, "Towards Open Set Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, 7 Jul. 2013. DOI: [10.1109/TPAMI.2012.256](https://doi.org/10.1109/TPAMI.2012.256).
- [98] O. Mayer, B. Hosler, and M. C. Stamm, "Open Set Video Camera Model Verification," *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 2962–2966, 2020, Barcelona, Spain. DOI: [10.1109/ICASSP40776.2020.9054261](https://doi.org/10.1109/ICASSP40776.2020.9054261).
- [99] W.-C. Yang, J. Jiang, and C.-H. Chen, "A fast source camera identification and verification method based on PRNU analysis for use in video forensic investigations," *Multimedia Tools and Applications*, vol. 80, no. 5, pp. 6617–6638, 2021. DOI: [10.1007/s11042-020-09763-z](https://doi.org/10.1007/s11042-020-09763-z).
- [100] E. Altinisik and H. T. Sencar, "Source Camera Verification for Strongly Stabilized Videos," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 643–657, 2021. DOI: [10.1109/TIFS.2020.3016830](https://doi.org/10.1109/TIFS.2020.3016830).
- [101] Y. Su, J. Xu, and B. Dong, "A Source Video Identification Algorithm Based on Motion Vectors," *Proceedings of the Second International Workshop on Computer Science and Engineering*, vol. 2, pp. 312–316, 2009, Qingdao, China. DOI: [10.1109/WCSE.2009.820](https://doi.org/10.1109/WCSE.2009.820).
- [102] S. Yahaya, A. T. S. Ho, and A. A. Wahab, "Advanced video camera identification using Conditional Probability Features," *Proceedings of the IET Conference on Image Processing*, pp. 1–5, 2012, London, UK. DOI: [10.1049/cp.2012.0426](https://doi.org/10.1049/cp.2012.0426).
- [103] M. Chen, J. Fridrich, M. Goljan, and J. Lukáš, "Source digital camcorder identification using sensor photo response non-uniformity," *Proceedings of the Security, Steganography, and Watermarking of Multimedia Contents IX*, vol. 6505, pp. 517–528, 2007. DOI: [10.1117/12.696519](https://doi.org/10.1117/12.696519).
- [104] L. J. G. Villalba, A. L. S. Orozco, R. R. López, and J. H. Castro, "Identification of smartphone brand and model via forensic video analysis," *Expert Systems with Applications*, vol. 55, pp. 59–69, 2016. DOI: [10.1016/j.eswa.2016.01.025](https://doi.org/10.1016/j.eswa.2016.01.025).
- [105] E. Altinisik, K. Tasdemir, and H. T. Sencar, "Mitigation of H.264 and H.265 Video Compression for Reliable PRNU Estimation," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1557–1571, 2020. DOI: [10.1109/TIFS.2019.2945190](https://doi.org/10.1109/TIFS.2019.2945190).
- [106] P. Ferrara, M. Iuliani, and A. Piva, "PRNU-Based Video Source Attribution: Which Frames Are You Using?" *Journal of Imaging*, vol. 8, no. 3, p. 57, 2022. DOI: [10.3390/jimaging8030057](https://doi.org/10.3390/jimaging8030057).

- [107] B. Hosler, O. Mayer, B. Bayar, X. Zhao, C. Chen, J. A. Shackelford, and M. C. Stamm, “A Video Camera Model Identification System Using Deep Learning and Fusion,” *Proceedings of the 2019 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 8271–8275, 2019, Brighton, UK. DOI: [10.1109/ICASSP.2019.8682608](https://doi.org/10.1109/ICASSP.2019.8682608).
- [108] D. Cozzolino, G. Poggi, and L. Verdoliva, “Extracting camera-based fingerprints for video forensics,” *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2019, Long Beach, CA, USA.
- [109] D. Timmerman, S. Bennabhaktula, E. Alegre, and G. Azzopardi, “Video Camera Identification from Sensor Pattern Noise with a Constrained ConvNet,” *arXiv preprint arXiv:2012.06277*, 2020. DOI: [10.48550/arXiv.2012.06277](https://doi.org/10.48550/arXiv.2012.06277).
- [110] D. Dal Cortivo, S. Mandelli, P. Bestagini, and S. Tubaro, “CNN-Based Multi-Modal Camera Model Identification on Video Sequences,” *Journal of Imaging*, vol. 7, no. 8, p. 135, 2021. DOI: [10.3390/jimaging7080135](https://doi.org/10.3390/jimaging7080135).
- [111] D. Vazquez-Padin, M. Fontani, T. Bianchi, P. Comesana, A. Piva, and M. Barni, “Detection of video double encoding with GOP size estimation,” *Proceedings of the IEEE International Workshop on Information Forensics and Security*, pp. 151–156, 2012, Costa Adeje, Spain. DOI: [10.1109/WIFS.2012.6412641](https://doi.org/10.1109/WIFS.2012.6412641).
- [112] Q. Xu, X. Jiang, T. Sun, P. He, S. Wang, and B. Li, “Relocated I-Frames Detection in H.264 Double Compressed Videos Based on Genetic-CNN,” *The Proceedings of the Asia-Pacific Signal and Information Processing Association Annual Summit and Conference*, pp. 710–716, 2018, Honolulu, HI, USA. DOI: [10.23919/APSIPA.2018.8659519](https://doi.org/10.23919/APSIPA.2018.8659519).
- [113] P. Bestagini, S. Milani, M. Tagliasacchi, and S. Tubaro, “Codec and GOP Identification in Double Compressed Videos,” *IEEE Transactions on Image Processing*, vol. 25, no. 5, pp. 2298–2310, 2016. DOI: [10.1109/TIP.2016.2541960](https://doi.org/10.1109/TIP.2016.2541960).
- [114] P. He, X. Jiang, T. Sun, S. Wang, B. Li, and Y. Dong, “Frame-wise detection of relocated I-frames in double compressed H. 264 videos based on convolutional neural network,” *Journal of Visual Communication and Image Representation*, vol. 48, pp. 149–158, 2017. DOI: [10.1016/j.jvcir.2017.06.010](https://doi.org/10.1016/j.jvcir.2017.06.010).
- [115] H. Yao, S. Song, C. Qin, Z. Tang, and X. Liu, “Detection of double-compressed H. 264/AVC video incorporating the features of the string of data bits and skip macroblocks,” *Symmetry*, vol. 9, no. 12, p. 313, 2017. DOI: [10.3390/sym9120313](https://doi.org/10.3390/sym9120313).
- [116] D. Vázquez-Padín, M. Fontani, D. Shullani, F. Pérez-González, A. Piva, and M. Barni, “Video Integrity Verification and GOP Size Estimation Via Generalized Variation of Prediction Footprint,” *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1815–1830, 2020. DOI: [10.1109/TIFS.2019.2951313](https://doi.org/10.1109/TIFS.2019.2951313).

- [117] G. Mahfoudi, F. Retraint, F. Morain-Nicolier, and M. M. Pic, “Statistical H.264 Double Compression Detection Method Based on DCT Coefficients,” *IEEE Access*, vol. 10, pp. 4271–4283, 2022. DOI: [10.1109/ACCESS.2022.3140588](https://doi.org/10.1109/ACCESS.2022.3140588).
- [118] S. Verde, L. Bondi, P. Bestagini, S. Milani, G. Calvagno, and S. Tubaro, “Video Codec Forensics Based on Convolutional Neural Networks,” *Proceedings of the IEEE International Conference on Image Processing*, pp. 530–534, 2018, Athens, Greece. DOI: [10.1109/ICIP.2018.8451143](https://doi.org/10.1109/ICIP.2018.8451143).
- [119] G. Sullivan and T. Wiegand, “Video Compression - From Concepts to the H.264/AVC Standard,” *Proceedings of the IEEE*, vol. 93, no. 1, pp. 18–31, 2005. DOI: [10.1109/JPROC.2004.839617](https://doi.org/10.1109/JPROC.2004.839617).
- [120] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, “Swin transformer: Hierarchical vision transformer using shifted windows,” *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10 012–10 022, 2021, Nashville, CA, USA. DOI: [10.1109/ICCV48922.2021.00986](https://doi.org/10.1109/ICCV48922.2021.00986).
- [121] Z. Liu, J. Ning, Y. Cao, Y. Wei, Z. Zhang, S. Lin, and H. Hu, “Video swin transformer,” *arXiv preprint arXiv:2106.13230*, 2021. DOI: [10.48550/arXiv.2106.13230](https://doi.org/10.48550/arXiv.2106.13230).
- [122] P. Verma and J. Berger, “Audio transformers: Transformer architectures for large scale audio understanding. adieu convolutions,” *arXiv preprint arXiv:2105.00335*, 2021. DOI: [10.48550/arXiv.2105.00335](https://doi.org/10.48550/arXiv.2105.00335).
- [123] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. DOI: [10.1162/neco.1997.9.8.1735](https://doi.org/10.1162/neco.1997.9.8.1735).
- [124] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, “On the properties of neural machine translation: Encoder-decoder approaches,” *arXiv preprint arXiv:1409.1259*, 2014. DOI: [10.48550/arXiv.1409.1259](https://doi.org/10.48550/arXiv.1409.1259).
- [125] A. Zeyer, P. Bahar, K. Irie, R. Schlüter, and H. Ney, “A Comparison of Transformer and LSTM Encoder Decoder Models for ASR,” *Proceedings of the IEEE Automatic Speech Recognition and Understanding Workshop*, pp. 8–15, 2019, Sentosa, Singapore. DOI: [10.1109/ASRU46091.2019.9004025](https://doi.org/10.1109/ASRU46091.2019.9004025).
- [126] O. Mayer and M. C. Stamm, “Forensic Similarity for Digital Images,” *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1331–1346, 2020. DOI: [10.1109/TIFS.2019.2924552](https://doi.org/10.1109/TIFS.2019.2924552).
- [127] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *arXiv preprint arXiv:1301.3781*, 2013. DOI: [10.48550/arXiv.1301.3781](https://doi.org/10.48550/arXiv.1301.3781).
- [128] G. Valenzise, M. Tagliasacchi, and S. Tubaro, “Estimating QP and Motion Vectors in H.264/AVC Video from Decoded Pixels,” *Proceedings of the 2nd ACM Workshop on Multimedia in Forensics, Security and Intelligence*, pp. 89–92, 2010, Firenze, Italy. DOI: [10.1145/1877972.1877995](https://doi.org/10.1145/1877972.1877995).

- [129] C. Sun, A. Myers, C. Vondrick, K. Murphy, and C. Schmid, “VideoBERT: A Joint Model for Video and Language Representation Learning,” *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 7463–7472, 2019, Seoul, Korea. doi: [10.1109/ICCV.2019.00756](https://doi.org/10.1109/ICCV.2019.00756).
- [130] J. Huang and C. Ling, “Using AUC and accuracy in evaluating learning algorithms,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 3, pp. 299–310, 2005. doi: [10.1109/TKDE.2005.50](https://doi.org/10.1109/TKDE.2005.50).
- [131] G. Hripcsak and A. S. Rothschild, “Agreement, the f-measure, and reliability in information retrieval,” *Journal of the American Medical Informatics Association*, vol. 12, no. 3, pp. 296–298, 2005. doi: [10.1197/jamia.M1733](https://doi.org/10.1197/jamia.M1733).
- [132] D. Shullani, M. Fontani, M. Iuliani, O. Al Shaya, and A. Piva, “VISION: A video and image dataset for source identification,” *EURASIP Journal on Information Security*, vol. 15, 2017. doi: [10.1186/s13635-017-0067-2](https://doi.org/10.1186/s13635-017-0067-2).
- [133] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *arXiv preprint arXiv:1512.03385*, 2015. doi: [10.48550/arXiv.1512.03385](https://doi.org/10.48550/arXiv.1512.03385).
- [134] International Organization for Standardization, *ISO/IEC 14496–14:2020–Information Technology–Coding of Audio-visual Objects–Part 14: MP4 File Format*, <https://www.iso.org/standard/79110.html>, 2020.
- [135] M. Iuliani, D. Shullani, M. Fontani, S. Meucci, and A. Piva, “A Video Forensic Framework for the Unsupervised Analysis of MP4-Like File Container,” *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 3, pp. 635–645, 2019. doi: [10.1109/TIFS.2018.2859760](https://doi.org/10.1109/TIFS.2018.2859760).
- [136] D. Güera, S. Baireddy, P. Bestagini, S. Tubaro, and E. J. Delp, “We Need No Pixels: Video Manipulation Detection Using Stream Descriptors,” *arXiv preprint arXiv:1906.08743*, 2019.
- [137] P. Yang, D. Baracchi, M. Iuliani, D. Shullani, R. Ni, Y. Zhao, and A. Piva, “Efficient Video Integrity Analysis Through Container Characterization,” *IEEE Journal of Selected Topics in Signal Processing*, vol. 14, no. 5, pp. 947–954, 2020. doi: [10.1109/JSTSP.2020.3008088](https://doi.org/10.1109/JSTSP.2020.3008088).
- [138] E. Altinisik, H. T. Sencar, and D. Tabaa, “Video Source Characterization Using Encoding and Encapsulation Characteristics,” *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 3211–3224, 2022.
- [139] K. Bhagtani, A. K. S. Yadav, E. R. Bartusiak, Z. Xiang, R. Shao, S. Baireddy, and E. J. Delp, “An Overview of Recent Work in Media Forensics: Methods and Threats,” *arXiv preprint arXiv:2204.12067*, 2022.

- [140] S. Milani, M. Fontani, P. Bestagini, M. Barni, A. Piva, M. Tagliasacchi, and S. Tubaro, “An Overview on Video Forensics,” *APSIPA Transactions on Signal and Information Processing*, vol. 1, e2, 2012.
- [141] L. Verdoliva, “Media Forensics and DeepFakes: An Overview,” *IEEE Journal of Selected Topics in Signal Processing*, vol. 14, no. 5, pp. 910–932, 2020. doi: [10.1109/JSTSP.2020.3002101](https://doi.org/10.1109/JSTSP.2020.3002101).
- [142] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, “Support Vector Machines,” *IEEE Intelligent Systems and Their Applications*, vol. 13, no. 4, pp. 18–28, 1998.
- [143] J. R. Quinlan, “Induction of Decision Trees,” *Machine learning*, vol. 1, pp. 81–106, 1986.
- [144] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, “Graph Attention Networks,” *arXiv preprint arXiv:1710.10903*, 2017.
- [145] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, “Heterogeneous Graph Attention Network,” *Proceedings of the 2019 World Wide Web Conference*, pp. 2022–2032, 2019, San Francisco, CA, USA.
- [146] D. W. Otter, J. R. Medina, and J. K. Kalita, “A Survey of the Usages of Deep Learning for Natural Language Processing,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 2, pp. 604–624, 2020.
- [147] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” *arXiv preprint arXiv:1810.04805*, 2018.
- [148] A. Thawani, J. Pujara, P. A. Szekely, and F. Ilievski, “Representing Numbers in NLP: a Survey and a Vision,” *arXiv preprint arXiv:2103.13136*, 2021.
- [149] Y. Gong, C.-I. Lai, Y.-A. Chung, and J. Glass, “SSAST: Self-supervised Audio Spectrogram Transformer,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 10, pp. 10 699–10 709, 2022, Virtual.
- [150] D. Hendrycks, M. Mazeika, S. Kadavath, and D. Song, “Using Self-Supervised Learning Can Improve Model Robustness and Uncertainty,” *Advances in Neural Information Processing Systems*, vol. 32, 2019.
- [151] I. Misra and L. v. d. Maaten, “Self-supervised Learning of Pretext-invariant Representations,” *Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6707–6717, 2020, Virtual.
- [152] R. Zhang, P. Isola, and A. A. Efros, “Split-brain Autoencoders: Unsupervised Learning by Cross-channel Prediction,” *Proceedings of the 2017 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1058–1067, 2017, Honolulu, HI, USA.

- [153] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-scale Image Recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
- [154] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016. DOI: [10.1109/CVPR.2016.90](https://doi.org/10.1109/CVPR.2016.90).
- [155] T. N. Kipf and M. Welling, “Semi-supervised Classification with Graph Convolutional Networks,” *arXiv preprint arXiv:1609.02907*, 2016.
- [156] J. Lee, I. Lee, and J. Kang, “Self-attention Graph Pooling,” *Proceedings of 2019 International Conference on Machine Learning*, pp. 3734–3743, 2019, Long Beach, CA, USA.
- [157] G. Biau and E. Scornet, “A Random Forest Guided Tour,” *Test*, vol. 25, pp. 197–227, 2016.
- [158] D. M. W. Powers, “Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation,” *Journal of Machine Learning Technologies*, vol. 2, no. 1, pp. 37–63, 2011.
- [159] H. Guan, M. Kozak, E. Robertson, Y. Lee, A. N. Yates, A. Delgado, D. Zhou, T. Kheyrkhah, J. Smith, and J. Fiscus, “MFC datasets: Large-scale Benchmark Datasets for Media Forensic Challenge Evaluation,” *Proceedings of the 2019 IEEE Winter Applications of Computer Vision Workshops*, pp. 63–72, 2019, Waikoloa, HI, USA.
- [160] I. Loshchilov and F. Hutter, “Decoupled Weight Decay Regularization,” *arXiv preprint arXiv:1711.05101*, 2017.
- [161] Q. Wang, Y. Ma, K. Zhao, and Y. Tian, “A Comprehensive Survey of Loss Functions in Machine Learning,” *Annals of Data Science*, vol. 9, no. 2, pp. 187–212, 2022.
- [162] A. Tharwat, “Classification Assessment Methods,” *Applied Computing and Informatics*, vol. 17, no. 1, pp. 168–192, 2021.
- [163] International Organization for Standardization, *ISO/IEC 13818-3:1995 - Information technology – Generic coding of moving pictures and associated audio information – Part 3: Audio*, 1995. [Online]. Available: <https://www.iso.org/standard/22991.html>.
- [164] International Organization for Standardization, *ISO/IEC 13818-7:1997 Information technology - Generic Coding of Moving Pictures and Associated Audio Information - Part 7: Advanced Audio Coding (AAC)*, 1997. [Online]. Available: <https://www.iso.org/standard/25040.html>.
- [165] M. Zakariah, M. K. Khan, and H. Malik, “Digital Multimedia Audio Forensics: Past, Present and Future,” *Multimedia tools and applications*, vol. 77, pp. 1009–1040, 2018. DOI: [10.1007/s11042-016-4277-2](https://doi.org/10.1007/s11042-016-4277-2).

- [166] T. Bianchi, A. De Rosa, M. Fontani, G. Rocciolo, and A. Piva, “Detection and Classification of Double Compressed MP3 Audio Tracks,” *Proceedings of the First ACM Workshop on Information Hiding and Multimedia Security*, 159–164, 2013, Montpellier, France. doi: [10.1145/2482513.2482523](https://doi.org/10.1145/2482513.2482523).
- [167] Z. Xiang, P. Bestagini, S. Tubaro, and E. J. Delp, “Forensic Analysis and Localization of Multiply Compressed MP3 Audio Using Transformers,” *Proceedings of the 2022 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 2929–2933, 2022, Singapore. doi: [10.1109/ICASSP43922.2022.9747639](https://doi.org/10.1109/ICASSP43922.2022.9747639).
- [168] A. V. Oppenheim, “Speech Spectrograms Using the Fast Fourier Transform,” *IEEE Spectrum*, vol. 7, no. 8, pp. 57–62, 1970. doi: [10.1109/MSPEC.1970.5213512](https://doi.org/10.1109/MSPEC.1970.5213512).
- [169] F. Zheng, G. Zhang, and Z. Song, “Comparison of Different Implementations of MFCC,” *Journal of Computer science and Technology*, vol. 16, pp. 582–589, 2001. doi: [10.1007/BF02943243](https://doi.org/10.1007/BF02943243).
- [170] J. C. Brown, “Calculation of a Constant Q Spectral Transform,” *The Journal of the Acoustical Society of America*, vol. 89, no. 1, pp. 425–434, 1991. doi: [10.1121/1.400476](https://doi.org/10.1121/1.400476).
- [171] M. M. Kabir, M. F. Mridha, J. Shin, I. Jahan, and A. Q. Ohi, “A Survey of Speaker Recognition: Fundamental Theories, Recognition Methods and Opportunities,” *IEEE Access*, vol. 9, pp. 79 236–79 263, 2021. doi: [10.1109/ACCESS.2021.3084299](https://doi.org/10.1109/ACCESS.2021.3084299).
- [172] P. Singh, G. Saha, and M. Sahidullah, “Non-linear Frequency Warping Using Constant-Q Transformation for Speech Emotion Recognition,” *Proceedings of the 2021 International Conference on Computer Communication and Informatics*, pp. 1–6, 2021. doi: [10.1109/ICCCI50826.2021.9402569](https://doi.org/10.1109/ICCCI50826.2021.9402569).
- [173] A. Nautsch, X. Wang, N. Evans, T. H. Kinnunen, V. Vestman, M. Todisco, H. Delgado, M. Sahidullah, J. Yamagishi, and K. A. Lee, “ASVspoof 2019: Spoofing Countermeasures for the Detection of Synthesized, Converted and Replayed Speech,” *IEEE Transactions on Biometrics, Behavior, and Identity Science*, vol. 3, no. 2, pp. 252–265, 2021. doi: [10.1109/TBIOM.2021.3059479](https://doi.org/10.1109/TBIOM.2021.3059479).
- [174] D. Ghosal and M. H. Kolekar, “Music Genre Recognition Using Deep Neural Networks and Transfer Learning,” *Proceedings of Interspeech 2018*, pp. 2087–2091, 2018. doi: [10.21437/Interspeech.2018-2045](https://doi.org/10.21437/Interspeech.2018-2045).
- [175] V. Bansal, G. Pahwa, and N. Kannan, “Cough Classification for COVID-19 Based on Audio MFCC Features Using Convolutional Neural Networks,” *Proceedings of the 2020 IEEE International Conference on Computing, Power and Communication Technologies*, pp. 604–608, 2020. doi: [10.1109/GUCON48875.2020.9231094](https://doi.org/10.1109/GUCON48875.2020.9231094).

- [176] E. Grinstein, N. Q. K. Duong, A. Ozerov, and P. Pérez, “Audio style transfer,” *Proceedings of the 2018 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 586–590, 2018. DOI: [10.1109/ICASSP.2018.8461711](https://doi.org/10.1109/ICASSP.2018.8461711).
- [177] J. Yamagishi, X. Wang, M. Todisco, M. Sahidullah, J. Patino, A. Nautsch, X. Liu, K. A. Lee, T. Kinnunen, N. Evans, *et al.*, “ASVspoof 2021: Accelerating Progress in Spoofed and Deepfake Speech Detection,” *arXiv preprint arXiv:2109.00537*, 2021. DOI: <https://doi.org/10.48550/arXiv.2109.00537>.
- [178] R. Reimao and V. Tzerpos, “FoR: A Dataset for Synthetic Speech Detection,” *Proceedings of the 2019 International Conference on Speech Technology and Human-Computer Dialogue*, pp. 1–10, 2019, Timisoara, Romania. DOI: [10.1109/SPED.2019.8906599](https://doi.org/10.1109/SPED.2019.8906599).
- [179] Z. Almutairi and H. Elgibreen, “A Review of Modern Audio Deepfake Detection Methods: Challenges and Future Directions,” *Algorithms*, vol. 15, no. 5, p. 155, 2022. DOI: [10.3390/a15050155](https://doi.org/10.3390/a15050155).
- [180] E. R. Bartusiak and E. J. Delp, “Frequency Domain-based Detection of Generated Audio,” *Proceedings of IS&T International Symposium on Electronic Imaging: Media Watermarking, Security, and Forensics*, pp. 273–1 –273–7, 2021, Virtual. DOI: [10.2352/ISSN.2470-1173.2021.4.MWSF-273](https://doi.org/10.2352/ISSN.2470-1173.2021.4.MWSF-273).
- [181] H. Khalid, M. Kim, S. Tariq, and S. S. Woo, “Evaluation of an Audio-Video Multimodal Deepfake Dataset Using Unimodal and Multimodal Detectors,” *Proceedings of the 1st Workshop on Synthetic Multimedia – Audiovisual Deepfake Generation and Detection*, pp. 7–15, 2021, Virtual. DOI: [10.1145/3476099.3484315](https://doi.org/10.1145/3476099.3484315).
- [182] P. A. Ziabary and H. Veisi, “A Countermeasure Based on CQT Spectrogram for Deepfake Speech Detection,” *Proceedings of the 2021 7th International Conference on Signal Processing and Intelligent Systems*, pp. 1–5, 2021. DOI: [10.1109/ICSPIS54653.2021.9729387](https://doi.org/10.1109/ICSPIS54653.2021.9729387).
- [183] J. S. Jacaba, “Audio Compression Using Modified Discrete Cosine Transform: The MP3 Coding Standard,” Bachelor’s Thesis, University of the Philippines, Manila, Oct. 2001. [Online]. Available: https://www.math.utah.edu/~gustafso/s2016/2270/project-ideas/audio-mp3-compression-MDCT-jacaba_main.pdf.
- [184] Y. Wang and M. Viterbo, “Modified Discrete Cosine Transform: Its Implications for Audio Coding and Error Concealment,” *Journal of the Audio Engineering Society*, vol. 51, no. 1/2, pp. 52–61, 2003.
- [185] T. Painter and A. Spanias, “Perceptual Coding of Digital Audio,” *Proceedings of the IEEE*, vol. 88, no. 4, pp. 451–515, 2000. DOI: [10.1109/5.842996](https://doi.org/10.1109/5.842996).
- [186] J. Rothweiler, “Polyphase Quadrature Filters—A New Subband Coding Technique,” *Proceedings of the 1983 IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 1280–1283, 1983. DOI: [10.1109/ICASSP.1983.1172005](https://doi.org/10.1109/ICASSP.1983.1172005).

- [187] T. Nguyen, “Near-perfect-reconstruction Pseudo-QMF Banks,” *IEEE Transactions on Signal Processing*, vol. 42, no. 1, pp. 65–76, 1994. doi: [10.1109/78.258122](https://doi.org/10.1109/78.258122).
- [188] G. Schuller, *Filter Banks and Audio Coding: Compressing Audio Signals Using Python*. Springer Nature, 2020. doi: [10.1007/978-3-030-51249-1](https://doi.org/10.1007/978-3-030-51249-1).
- [189] H. Purwins, “Profiles of Pitch Classes Circularity of Relative Pitch and Key—Experiments, Models, Computational Music Analysis, and Perspectives,” Ph.D. dissertation, Technische Universität Berlin, 2005. [Online]. Available: <https://depositonce.tu-berlin.de/items/4cb9db3c-3ff1-4849-bb62-936df7fde7b3>.
- [190] J. S. Sobolewski, “Data Transmission Media,” in *Encyclopedia of Physical Science and Technology (Third Edition)*, R. A. Meyers, Ed., New York: Academic Press, 2003, pp. 277–303, ISBN: 978-0-12-227410-7. doi: <https://doi.org/10.1016/B0-12-227410-5/00165-4>.
- [191] lieff, *Minimp3: minimalistic mp3 decoder single header library*, 2018. [Online]. Available: <https://github.com/lieff/minimp3>.
- [192] TorchAudio Contributors, *Torchaudio documentation*, 2023. [Online]. Available: <https://pytorch.org/audio/master/index.html>.
- [193] R. Reimao and V. Tzerpos, “Synthetic Speech Detection Using Neural Networks,” *Proceedings of the 2021 International Conference on Speech Technology and Human-Computer Dialogue*, pp. 97–102, 2021. doi: [10.1109/SpeD53181.2021.9587406](https://doi.org/10.1109/SpeD53181.2021.9587406).
- [194] C. K. Wai, *Nnaudio 0.3.1*, 2023. [Online]. Available: <https://kinwaicheuk.github.io/nnAudio/index.html>.
- [195] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016, Las Vegas, NV, USA. doi: [10.1109/CVPR.2016.90](https://doi.org/10.1109/CVPR.2016.90).
- [196] M. Tan and Q. Le, “EfficientNetV2: Smaller Models and Faster Training,” *Proceedings of International Conference on Machine Learning*, vol. 139, pp. 10 096–10 106, 2021, Virtual.
- [197] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, “Aggregated Residual Transformations for Deep Neural Networks,” *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5987–5995, 2017, Honolulu, HI, USA. doi: [10.1109/CVPR.2017.634](https://doi.org/10.1109/CVPR.2017.634).
- [198] A. Howard, M. Sandler, B. Chen, W. Wang, L.-C. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang, H. Adam, and Q. Le, “Searching for MobileNetV3,” *Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision*, pp. 1314–1324, 2019. doi: [10.1109/ICCV.2019.00140](https://doi.org/10.1109/ICCV.2019.00140).

- [199] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge,” *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015. doi: [10.1007/s11263-015-0816-y](https://doi.org/10.1007/s11263-015-0816-y).
- [200] T. Fawcett, “An introduction to ROC analysis,” *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006. doi: [10.1016/j.patrec.2005.10.010](https://doi.org/10.1016/j.patrec.2005.10.010).
- [201] M. E. Schuckers, “Receiver operating characteristic curve and equal error rate,” in *Computational Methods in Biometric Authentication: Statistical Methods for Performance Evaluation*, London: Springer London, 2010, pp. 155–204, ISBN: 978-1-84996-202-5. doi: [10.1007/978-1-84996-202-5_5](https://doi.org/10.1007/978-1-84996-202-5_5).

VITA

Ziyue “Alan” Xiang is a Ph.D. candidate in School of Electrical and Computer Engineering at Purdue University, West Lafayette. He acquired Masters of Science degree in Computer Science at Syracuse University in 2020. He acquired Bachelor of Science degree in Information and Computing Science at Sun Yat-Sen University (Guangdong, China) in 2018. His bachelor thesis won the best undergraduate thesis award in Sun Yat-Sen University.

He started his Ph.D. program at Purdue University in 2020. Since then, he has been working on projects sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL). His research areas of interest includes media forensics, computer vision, machine learning, signal processing, video compression, and audio compression.

In 2023, he conducted an internship at Tencent America located in Palo Alto, CA, which focused on developing next generation international video coding standard beyond Versatile Video Coding (VVC).

He is a student member of IEEE, the IEEE Signal Processing Society, and the IEEE Computer Society.

PUBLICATION(S)

Publications Resulting From This Work

Conference Papers

- **Z. Xiang**, J. Horváth, S. Baireddy, P. Bestagini, S. Tubaro, and E. J. Delp, “Forensic Analysis of Video Files Using Metadata,” *Proceedings of the 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 1042–1051, Jun. 2021, Virtual. DOI: [10.1109/CVPRW53098.2021.00115](https://doi.org/10.1109/CVPRW53098.2021.00115)
- **Z. Xiang**, P. Bestagini, S. Tubaro, and E. J. Delp, “Forensic Analysis and Localization of Multiply Compressed MP3 Audio Using Transformers,” *Proceedings of the 2022 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 2929–2933, May 2022, Singapore. DOI: [10.1109/ICASSP43922.2022.9747639](https://doi.org/10.1109/ICASSP43922.2022.9747639)
- **Z. Xiang**, P. Bestagini, S. Tubaro, and E. J. Delp, “H4VDM: H.264 Video Device Matching,” *Pattern Recognition, Computer Vision, and Image Processing. ICPR 2022 International Workshops and Challenges*, pp. 300–319, Aug. 2022, Montreal, QC, Canada. DOI: [10.1007/978-3-031-37742-6_24](https://doi.org/10.1007/978-3-031-37742-6_24)
- **Z. Xiang**, A. K. S. Yadav, P. Bestagini, S. Tubaro, and E. J. Delp, “MTN: Forensic Analysis of MP4 Video Files Using Graph Neural Networks,” *Proceedings of the 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 963–972, Jun. 2023, Vancouver, BC, Canada. DOI: [10.1109/CVPRW59228.2023.00103](https://doi.org/10.1109/CVPRW59228.2023.00103)
- **Z. Xiang**, A. K. S. Yadav, S. Tubaro, P. Bestagini, and E. J. Delp, “Extracting Efficient Spectrograms From MP3 Compressed Speech Signals for Synthetic Speech Detection,” *Proceedings of the 2023 ACM Workshop on Information Hiding and Multimedia Security*, pp. 163–168, Jun. 2023, Chicago, IL, USA. DOI: [10.1145/3577163.3595104](https://doi.org/10.1145/3577163.3595104)

Other Publications Not Related to This Work

Book Chapters

- H. Hao, E. R. Bartusiak, D. Güera, D. Mas Montserrat, S. Baireddy, **Z. Xiang**, S. K. Yarlagadda, R. Shao, J. Horváth, J. Yang, F. Zhu, and E. J. Delp, “Deepfake Detection Using Multiple Data Modalities,” in *Handbook of Digital Face Manipulation and Detection: From DeepFakes to Morphing Attacks*, C. Rathgeb, R. Tolosana, R. Vera-Rodriguez, and C. Busch, Eds. Cham: Springer International Publishing, 2022, pp. 235–254. doi: [10.1007/978-3-030-87664-7_11](https://doi.org/10.1007/978-3-030-87664-7_11)

Conference Papers

- K. Bhagtani, A. K. S. Yadav, E. R. Bartusiak, **Z. Xiang**, R. Shao, S. Baireddy, and E. J. Delp, “An Overview of Recent Work in Multimedia Forensics,” pp. 324–329, Aug. 2022, Virtual
- A. K. Singh Yadav, **Z. Xiang**, E. R. Bartusiak, P. Bestagini, S. Tubaro, and E. J. Delp, “ASSD: Synthetic Speech Detection in the AAC Compressed Domain,” pp. 1–5, 2023, Rhodes Island, Greece. doi: [10.1109/ICASSP49357.2023.10095043](https://doi.org/10.1109/ICASSP49357.2023.10095043)
- J. Horváth, **Z. Xiang**, E. D. Cannas, P. Bestagini, S. Tubaro, and E. J. Delp, “Sat U-net: a Fusion Based Method for Forensic Splicing Localization in Satellite Images,” *Multimodal Image Exploitation and Learning 2022*, vol. 12100, S. S. Agaian, V. K. Asari, S. P. DelMarco, and S. A. Jassim, Eds., p. 1 210 002, 2022, Orlando, FL, USA. doi: [10.1117/12.2616150](https://doi.org/10.1117/12.2616150)
- K. Bhagtani, A. K. S. Yadav, **Z. Xiang**, P. Bestagini, and E. J. Delp, “FGSSAT: Unsupervised Fine-grain Attribution of Unknown Speech Synthesizers Using Transformer Networks,” *Proceedings of 2023 Asilomar Conference on Signals, Systems, and Computers*, Oct. 2023, Pacific Groove, CA, USA
- A. K. S. Yadav, K. Bhagtani, **Z. Xiang**, P. Bestagini, S. Tubaro, and E. J. Delp, “DSVAE: Interpretable Disentangled Representation for Synthetic Speech Detection,” *Proceedings of the 2023 IEEE International Conference on Machine Learning and Applications*, 2023, Jacksonville, FL, USA. doi: [10.48550/arXiv.2304.03323](https://doi.org/10.48550/arXiv.2304.03323)

- A. K. S. Yadav, **Z. Xiang**, K. Bhagtani, P. Bestagini, S. Tubaro, and E. J. Delp, “PS3DT: Synthetic Speech Detection Using Patched Spectrogram Transformer,” *Proceedings of the 2023 IEEE International Conference on Machine Learning and Applications*, 2023, Jacksonville, FL, USA

Preprint Papers

- K. Bhagtani, A. K. S. Yadav, E. R. Bartusiak, **Z. Xiang**, R. Shao, S. Baireddy, and E. J. Delp, “An Overview of Recent Work in Multimedia Forensics: Methods and threats,” *arXiv preprint arXiv:2204.12067*, 2022. DOI: [10.48550/arXiv.2204.12067](https://doi.org/10.48550/arXiv.2204.12067)
- A. O. Pellicer, A. K. S. Yadav, K. Bhagtani, **Z. Xiang**, Z. Pizlo, I. Gradus-Pizlo, and E. J. Delp, “Synthetic Echocardiograms Generation Using Diffusion Models,” *bioRxiv preprint bioRxiv:2023.11.11.566718*, 2023. DOI: [10.1101/2023.11.11.566718](https://doi.org/10.1101/2023.11.11.566718)
- **Z. Xiang** and D. E. Acuna, “Scientific image tampering detection based on noise inconsistencies: A method and datasets,” *arXiv preprint arXiv:2001.07799*, 2020. DOI: [10.48550/arXiv.2001.07799](https://doi.org/10.48550/arXiv.2001.07799)
- D. E. Acuna and **Z. Xiang**, “Estimating a Null Model of Scientific Image Reuse to Support Research Integrity Investigations,” *arXiv preprint arXiv:2003.00878*, 2020. DOI: [10.48550/arXiv.2003.00878](https://doi.org/10.48550/arXiv.2003.00878)